determining whether the one or more APIs to which the software application requires access includes a sensitive API;

determining whether the software application includes an authentic global signature; and

determining whether the software application includes an authentic digital signature and signature identification where the one or more APIs to which the software application requires access includes a sensitive API and the software application includes an authentic global signature; and

the step of denying the software application access to the one or more APIs comprises the steps of:

denying the software application access to the one or more APIs where the software application does not include an authentic global signature; and

denying the software application access to the sensitive API where the one or more APIs to which the software application requires access includes a sensitive API, the software application includes an authentic global signature, and the software application does not include an authentic digital signature and signature identifier required to access the sensitive API.

112. (New) A code signing system for controlling access to application programming interfaces (APIs) having signature identificaters by software applications, the code signing system comprising:

a verification system for authenticating digital signatures provided by the respective software applications to access the APIs where the signature identifications correspond with the signature identificaters of the respective APIs and where a digital signature for a software application is generated with a signature identification corresponding to a signature identificater to access at least one API; and

a control system for allowing access to at least one of the APIs where the digital signature provided by the software application is authenticated by the verification system.

113. (New) The code signing system of claim 112, wherein a virtual machine comprises the verification system and the control system.

114. (New) The code signing system of claim 113, wherein the virtual machine is a Java virtual machine installed on a mobile device.

115. (New) The code signing system of claim 112, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

116. (New) The code signing system of claim 112, wherein the code signing system is installed on a mobile device and the software application is a Java application for a mobile device.

117. (New) The code signing system of claim 112, wherein the digital signature and the signature identification of the software application are generated by a code signing authority.

118. (New) The code signing system of claim 112, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

119. (New) The code signing system of claim 112, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and the verification system uses a public signature key to authenticate the digital signature.

120. (New) The code signing system of claim 119, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

121. (New) The code signing system of claim 112, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

122. (New) The code signing system of claim 112, wherein the APIs provides access to at least one of one or more core functions of a mobile device, an operating system, and hardware on a mobile device.

123. (New) The code signing system of claim 112, wherein verification of a global digital signature provided by the software application is required for accessing any of the APIs.

124. (New) A method of controlling access to application programming interfaces (APIs) having signature identifiers by software applications, the method comprising:

authenticating digital signatures provided by the respective software applications to access the APIs where the signature identifications correspond with the signature identifiers of the respective APIs and where a digital signature for a software application is generated with a signature identification corresponding to a signature identifier to access at least one API; and

allowing access to at least one of the APIs where the digital signature provided by the software application is authenticated.

125. (New) The method of claim 124, wherein one digital signature and one signature identification are provided by the software application access a library of at least one of the APIs.

126. (New) The method of claim 124, wherein the digital signature and the signature identification of the software application are generated by a code signing authority.

127. (New) The method of claim 124, wherein the APIs access at least one of a cryptographic module that implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

128. (New) The method of claim 124, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and a public signature key is used to authenticate the digital signature.

129. (New) The method of claim 128, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the digital signature is authenticated by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

130. (New) The method of claim 124, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

131. (New) The method of claim 124, wherein the APIs provides access to at least one of one or more core functions of a mobile device, an operating system, and hardware on a mobile device.

132. (New) The method of claim 124, wherein verification of a global digital signature provided by the software application is required for accessing any of the APIs

133. (New) A management system for controlling access by software applications to application programming interfaces (APIs) having at least one signature identifier on a subset of a plurality of mobile devices, the management system comprising:

a code signing authority for providing digital signatures and signature identifications to software applications that require access to at least one of the APIs with a signature identifier on the subset of the plurality of mobile devices, where a digital signature for a software application is generated with a signature identification corresponding to a signature identifier, and the signature identifications provided to the software applications comprise those signature

identifications that correspond to the signature identifiers that are substantially only on the subset of the plurality of mobile devices; wherein each mobile device of the subset of the plurality of mobile devices comprises

a verification system for authenticating digital signatures provided by the respective software applications to access respective APIs where the digital identifications correspond to the digital identifiers of the respective APIs; and

a control system for allowing the respective software applications to access at least one of the APIs where the digital signatures provided by the respective software applications are authenticated by the verification system.

134. (New) The management system of claim 133, wherein a virtual machine comprises the verification system and the control system.

135. (New) The management system of claim 134, wherein the virtual machine is a Java virtual machine and the software applications are Java applications.

136. (New) The management system of claim 133, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

137. (New) The management system of claim 133, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

138. (New) The management system of claim 133, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and the verification system uses a public signature key to authenticate the digital signature.

139. (New) The management system of claim 138, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

140. (New) The management system of claim 133, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

141. (New) The management system of claim 133, wherein the subset of the plurality of mobile devices comprises mobile devices under the control of at least one of a corporation and a carrier.

142. (New) The management system of claim 133, wherein a global digital signature provided by the software application has to be authenticated before the software application is allowed access to any of the APIs on a mobile device of the subset of the plurality of mobile devices.

143. (New) A method of controlling access by software applications to application programming interfaces (APIs) having at least one signature identifier on a subset of a plurality of mobile devices, the method comprising:

generating digital signatures for software applications with signature identifications corresponding to respective signature identifiers of the APIs; and

providing the digital signatures and the signature identifications to software applications that require access to at least one of the APIs on the subset of the plurality of mobile devices, where the signature identifications provided to the software applications comprise those signature identifications that correspond to the signature identifiers that are substantially only on the subset of the plurality of mobile devices; wherein each mobile device of the subset of the plurality of mobile devices comprises

a verification system for authenticating digital signatures provided by the respective software applications to access respective APIs where the digital identifications correspond to the digital identifiers of the respective APIs; and

a control system for allowing the software application to access at least one of the APIs where the digital signature provided by the software application is authenticated by the verification system.

144. (New) The method of claim 143, wherein a virtual machine comprises the verification system and the control system.

145. (New) The method of claim 144, wherein the virtual machine is a Java virtual machine and the software applications are Java applications.

146. (New) The method of claim 143, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

147. (New) The method of claim 143, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

148. (New) The method of claim 143, wherein at least one of the digital signatures is generated using a private signature key under a signature scheme associated with a signature identification, and the verification system uses a public signature keys to authenticate said at least one of the digital signatures.

149. (New) The method of claim 148, wherein:

at least one of the digital signatures is generated by applying the private signature key to a hash of a software application under the signature scheme; and

the verification system authenticates said at least one of the digital signatures by generating a hash of the software application to obtain a generated hash, applying the public

signature key to said at least one of the digital signatures to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

150. (New) The method of claim 143, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

151. (New) The method of claim 143, wherein the subset of the plurality of mobile devices comprises mobile devices under the control of at least one of a corporation and a carrier.

152. (New) A mobile device for a subset of a plurality of mobile devices, the mobile device comprising:

an application platform having application programming interfaces (APIs);

a verification system for authenticating digital signatures and signature identifications provided by the respective software applications to access the APIs; and

a control system for allowing a software application to access at least one of the APIs where a digital signature provided by the software application is authenticated by the verification system;

wherein a code signing authority provides digital signatures and signature identifications to software applications that require access to at least one of the APIs such that the digital signature for the software application is generated according to a signature scheme of a signature identification, and wherein the signature identifications provided to the software applications comprise those signature identifications that are substantially only authorized to allow access on the subset of the plurality of mobile devices.

153. (New) The mobile device of claim 152, wherein a virtual machine comprises the verification system and the control system.

154. (New) The mobile device of claim 153, wherein the virtual machine is a Java virtual machine and the software application is a Java application.

155. (New) The mobile device of claim 152, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

156. (New) The mobile device of claim 152, wherein the APIs of the application platform access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

157. (New) The mobile device of claim 152, wherein the digital signature is generated using a private signature key under the signature scheme, and the verification system uses a public signature key to authenticate the digital signature.

158. (New) The mobile device of claim 157, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

159. (New) The mobile device of claim 152, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

160. (New) A method of controlling access to application programming interfaces (APIs) of an application platform of a mobile device for a subset of a plurality of mobile devices, the method comprising:

receiving digital signatures and signature identifications from software applications that require to access the APIs

authenticating the digital signatures and the signature identifications; and

allowing a software application to access at least one of the APIs where a digital signature provided by the software application is authenticated;

wherein a code signing authority provides the digital signatures and the signature identifications to the software applications that require access to at least one of the APIs such that the digital signature for the software application is generated according to a signature scheme of a signature identification, and wherein the signature identifications provided to the software applications comprise those signature identifications that are substantially only authorized to allow access on the subset of the plurality of mobile devices.

161. (New) The method of claim 160, wherein one digital signature and one signature identification is required for accessing each library of at least one of the APIs.

162. (New) The method of claim 160, wherein the APIs of the application platform access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

163. (New) The method of claim 160, wherein the digital signature is generated using a private signature key under the signature scheme, and a public signature key is used to authenticate the digital signature.

164. (New) The method of claim 163, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the digital signature is authenticated by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

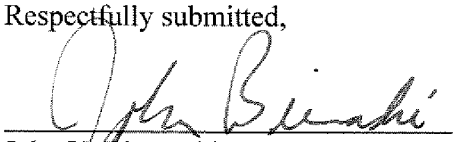
165. (New) The method of claim 160, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

REMARKS

This paper responds to the notice of non-compliant amendment mailed May 21, 2007. The examiner is invited to contact the undersigned in case there are any questions or comments.

Respectfully submitted,



John V. Biernacki
Reg. No. 40,511
Jones, Day
North Point
901 Lakeside Avenue
Cleveland, OH 44114-1190
(216) 586-7747

Electronic Acknowledgement Receipt	
EFS ID:	1811276
Application Number:	10381219
International Application Number:	
Confirmation Number:	9761
Title of Invention:	Software code signing system and method
First Named Inventor/Applicant Name:	David P Yach
Correspondence Address:	David B Cochran Jones Day North Point 901 Lakeside Avenue Cleveland OH 44114-1190 US - -
Filer:	Stephen D. Scanlon/Debra Pejeau
Filer Authorized By:	Stephen D. Scanlon
Attorney Docket Number:	555255012423
Receipt Date:	25-MAY-2007
Filing Date:	20-MAR-2003
Time Stamp:	11:27:25
Application Type:	U.S. National Stage under 35 USC 371

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)	Multi Part /.zip	Pages (if appl.)
1		10289USPCTResponse.pdf	731624	yes	23

	Multipart Description/PDF files in .zip description		
	Document Description	**Start**	**End**
	Preliminary Amendment	1	1
	Claims	2	22
	Applicant Arguments/Remarks Made in an Amendment	23	23

Warnings:	
Information:	
Total Files Size (in bytes):	731624

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/381,219	03/20/2003	David P Yach	555255012423	9761

7590 05/21/2007

David B Cochran
Jones Day
North Point
901 Lakeside Avenue
Cleveland, OH 44114-1190

EXAMINER	
AVERY, JEREMIAH L	
ART UNIT	PAPER NUMBER
2131	

MAIL DATE	DELIVERY MODE
05/21/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of Non-Compliant Amendment (37 CFR 1.121)	Application No.	Applicant(s)	
	10381219	YACH ET AL.	
	Examiner	Art Unit	
	Jeremiah Avery	2131	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

The amendment document filed on *03 May 2007* is considered non-compliant because it has failed to meet the requirements of 37 CFR 1.121 or 1.4. In order for the amendment document to be compliant, correction of the following item(s) is required.

THE FOLLOWING MARKED (X) ITEM(S) CAUSE THE AMENDMENT DOCUMENT TO BE NON-COMPLIANT:

- ☐ 1. Amendments to the specification:
  - ☐ A. Amended paragraph(s) do not include markings.
  - ☐ B. New paragraph(s) should not be underlined.
  - ☐ C. Other _____.
- ☐ 2. Abstract:
  - ☐ A. Not presented on a separate sheet. 37 CFR 1.72.
  - ☐ B. Other _____.
- ☐ 3. Amendments to the drawings:
  - ☐ A. The drawings are not properly identified in the top margin as "Replacement Sheet," "New Sheet," or "Annotated Sheet" as required by 37 CFR 1.121(d).
  - ☐ B. The practice of submitting proposed drawing correction has been eliminated. Replacement drawings showing amended figures, without markings, in compliance with 37 CFR 1.84 are required.
  - ☐ C. Other _____.
- ☐ 4. Amendments to the claims:
  - ☐ A. A complete listing of all of the claims is not present.
  - ☐ B. The listing of claims does not include the text of all pending claims (including withdrawn claims)
  - ☐ C. Each claim has not been provided with the proper status identifier, and as such, the individual status of each claim cannot be identified. Note: the status of every claim must be indicated after its claim number by using one of the following status identifiers: (Original), (Currently amended), (Canceled), (Previously presented), (New), (Not entered), (Withdrawn) and (Withdrawn-currently amended).
  - ☐ D. The claims of this amendment paper have not been presented in ascending numerical order.
  - ☐ E. Other: _____.
- ☒ 5. Other (e.g., the amendment is unsigned or not signed in accordance with 37 CFR 1.4):
  Claims section should start on a separate page from page 1.

For further explanation of the amendment format required by 37 CFR 1.121, see MPEP § 714.

TIME PERIODS FOR FILING A REPLY TO THIS NOTICE:

1. Applicant is given **no new time period** if the non-compliant amendment is an after-final amendment, an amendment filed after allowance, or a drawing submission (only). If applicant wishes to resubmit the non-compliant after-final amendment with corrections, the **entire corrected amendment** must be resubmitted.

2. Applicant is given **one month**, or thirty (30) days, whichever is longer, from the mail date of this notice to supply the correction, if the non-compliant amendment is one of the following: a preliminary amendment, a non-final amendment (including a submission for a request for continued examination (RCE) under 37 CFR 1.114), a supplemental amendment filed within a suspension period under 37 CFR 1.103(a) or (c), and an amendment filed in response to a *Quayle* action. If any of above boxes 1. to 4. are checked, the correction required is only the **corrected section** of the non-compliant amendment in compliance with 37 CFR 1.121.

   Extensions of time are available under 37 CFR 1.136(a) only if the non-compliant amendment is a non-final amendment or an amendment filed in response to a *Quayle* action.

   Failure to timely respond to this notice will result in:
   Abandonment of the application if the non-compliant amendment is a non-final amendment or an amendment filed in response to a *Quayle* action; or
   Non-entry of the amendment if the non-compliant amendment is a preliminary amendment or supplemental amendment.

Legal Instruments Examiner (LIE), if applicable _____ Telephone No. _____

U.S. Patent and Trademark Office Part of Paper No.

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In the application of	:	David P. Yach; Michael S. Brown; Herbert A. Little
Internat'l. Appl'n. No.	:	PCT/CA01/01344
Internat'l. Filing Date	:	09/20/2001
U.S. Serial No.	:	10/381,219
U.S. Filing Date	:	03/20/2003
Priority Date Claimed:		09/21/2000
Title	:	Software Code Signing System And Method
Art Unit	:	2131
Examiner	:	J. Avery
Docket No.	:	555255012423

Commissioner for Patents
Washington, D.C. 20231

Preliminary Amendment

This paper responds to the notice of non-compliant amendment mailed April 3, 2007. Any fees due should be charged to Jones Day Deposit Account No. 501432, ref: 555255-012423.

Prior to taking up this case for initial examination, please amend the application as follows.

The Claims

Please cancel original claims 1-56.

Please add the following new claims 57-165.

57. (New) A code signing system for operation in conjunction with a software application having a digital signature and a signature identification, where the digital signature is associated with the signature identification, comprising:

an application platform;

an application programming interface (API) having an associated signature identifier, the API is configured to link the software application with the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application where the signature identifier corresponds to the signature identification.

58. (New) The code signing system of claim 57, wherein the virtual machine denies the software application access to the API if the digital signature is not authenticated.

59. (New) The code signing system of claim 57, wherein the virtual machine purges the software application if the digital signature is not authenticated.

60. (New) The code signing system of claim 57, wherein the code signing system is installed on a mobile device.

61. (New) The code signing system of claim 57, wherein the digital signature is generated by a code signing authority.

62. (New) A code signing system for operation in conjunction with a software application having a digital signature and a signature identification where the digital signature is associated with the signature identification, comprising:

an application platform;

a plurality of application programming interfaces (APIs) associated with a signature identifier, each configured to link the software application with a resource on the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the APIs by the software application where the signature identification corresponds to the signature identifier,

wherein the virtual machine verifies the authenticity of the digital signature in order to control access to the plurality of APIs by the software application.

63. (New) The code signing system of claim 62, wherein the plurality of APIs are included in an API library.

64. (New) The code signing system of claim 62, wherein one or more of the plurality of APIs is classified as sensitive and having an associated signature identifier, and wherein the virtual machine uses the digital signature and the signature identification to control access to the sensitive APIs.

65. (New) The code signing system of claim 64, wherein the code signing system operates in conjunction with a plurality of software applications, wherein one or more of the plurality of software applications has a digital signature and a signature identification, and wherein the virtual machine verifies the authenticity of the digital signature of each of the one or more of the plurality of software applications, where the signature identification corresponds to the signature identifier of the respective sensitive APIs, in order to control access to the sensitive APIs by each of the plurality of software applications.

66. (New) The code signing system of claim 62, wherein the resource on the application platform comprises a wireless communication system.

67. (New) The code signing system of claim 62, wherein the resource on the application platform comprises a cryptographic module which implements cryptographic algorithms.

68. (New) The code signing system of claim 62, wherein the resource on the application platform comprises a data store.

69. (New) The code signing system of claim 62, wherein the resource on the application platform comprises a user interface (UI).

70. (New) The code signing system of claim 57, further comprising:

a plurality of API libraries, each of the plurality of API libraries includes a plurality of APIs, wherein the virtual machine controls access to the plurality of API libraries by the software application.

71. (New) The code signing system of claim 70, wherein at least one of the plurality of API libraries is classified as sensitive;

wherein access to a sensitive API library requires a digital signature associated with a signature identification where the signature identification corresponds to a signature identifier associated with the sensitive API library;

wherein the software application includes at least one digital signature and at least one associated signature identification for accessing sensitive API libraries; and

wherein the virtual machine authenticates the software application for accessing the sensitive API library by verifying the one digital signature included in the software application that has a signature identification corresponding to the signature identifier of the sensitive API library.

72. (New) The code signing system of claim 57, wherein the digital signature is generated using a private signature key, and the virtual machine uses a public signature key to verify the authenticity of the digital signature.

73. (New) The code signing system of claim 72, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application; and

the virtual machine verifies the authenticity of the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and comparing the generated hash with the recovered hash.

74. (New) The code signing system of claim 60, wherein the API further comprises:

a description string that is displayed by the mobile device when the software application attempts to access the API.

75. (New) The code signing system of claim 57, wherein the application platform comprises an operating system.

76 (New) The code signing system of claim 57, wherein the application platform comprises one or more core functions of a mobile device.

77. (New) The code signing system of claim 57, wherein the application platform comprises hardware on a mobile device.

78. (New) The code signing system of claim 57, wherein the hardware comprises a subscriber identity module (SIM) card.

79. (New) The code signing system of claim 57, wherein the software application is a Java application for a mobile device.

80. (New) The code signing system of claim 57, wherein the API interfaces with a cryptographic routine on the application platform.

81. (New) The code signing system of claim 57, wherein the API interfaces with a proprietary data model on the application platform.

82. (New) The code signing system of claim 57, wherein the virtual machine is a Java virtual machine installed on a mobile device.

83. (New) A method of controlling access to sensitive application programming interfaces on a mobile device, comprising the steps of:

loading a software application on the mobile device that requires access to a sensitive application programming interface (API) having a signature identifier;

determining whether the software application includes a digital signature and a signature identification; and

denying the software application access to the sensitive API where the signature identification does not correspond with the signature identifier.

84. (New) The method of claim 83, comprising the additional step of:

purging the software application from the mobile device where the signature identification does not correspond with the signature identifier.

85. (New) The method of claim 83, wherein the digital signature and the signature identification are generated by a code signing authority.

86. (New) The method of claim 83, comprising the additional steps of:

verifying the authenticity of the digital signature where the signature identification corresponds with the signature identifier.; and

denying the software application access to the sensitive API where the digital signature is not authenticated.

87. (New) The method of claim 86, comprising the additional step of:

purging the software application from the mobile device where the digital signature is not authenticated.

88. (New) The method of claim 86, wherein the digital signature is generated by applying a private signature key to a hash of the software application, and wherein the step of verifying the authenticity of the digital signature is performed by a method comprising the steps of:

storing a public signature key that corresponds to the private signature key on the mobile device;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

89. (New) The method of claim 88, wherein the digital signature is generated by calculating a hash of the software application and applying the private signature key.

90. (New) The method of claim 83, comprising the additional step of:

displaying a description string that notifies a user of the mobile device that the software application requires access to the sensitive API.

91. (New) The method of claim 90, comprising the additional step of:

receiving a command from the user granting or denying the software application access to the sensitive API.

92. (New) A method of controlling access to an application programming interface (API) having a signature identifier on a mobile device by a software application created by a software developer, comprising the steps of:

receiving the software application from the software developer;

determining whether the software application satisfies at least one criterion;

appending a digital signature and a signature identification to the software application where the software application satisfies at least one criterion;;

verifying the authenticity of the digital signature appended to the software application where the signature identification corresponds with the signature identifier; and

providing access to the API to software applications where the digital signature is authenticated.

93. (New) The method of claim 92, wherein the step of determining whether the software application satisfies at least one criterion is performed by a code signing authority.

94. (New) The method of claim 92, wherein the step of appending the digital signature and the signature identification to the software application includes generating the digital signature comprising the steps of:

calculating a hash of the software application; and

applying a signature key to the hash of the software application to generate the digital signature.

95. (New) The method of claim 94, wherein the hash of the software application is calculated using the Secure Hash Algorithm (SHA1).

96. (New) The method of claim 94, wherein the step of verifying the authenticity of the digital signature comprises the steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash; and

authenticating the digital signature by comparing the calculated hash with the recovered hash.

97. (New) The method of claim 96, comprising the further step of denying the software application access to the API where the digital signature is not authenticated.

98. (New) The method of claim 96, wherein the signature key is a private signature key and the corresponding signature key is a public signature key.

99. (New) A method of controlling access to a sensitive application programming interface (API) having a signature identifier on a mobile device, comprising the steps of:

registering one or more software developers that are trusted to develop software applications which access the sensitive API;

receiving a hash of a software application;

determining whether the hash was sent by a registered software developer; and

generating a digital signature using the hash of the software application and a signature identification corresponding to the signature identifier where the hash was sent by the registered software developer;

wherein

the digital signature and the signature identification are appended to the software application; and

the mobile device verifies the authenticity of the digital signature in order to control access to the sensitive API by the software application where the signature identification corresponds with the signature identifier.

100. (New) The method of claim 99, wherein the step of generating the digital signature is performed by a code signing authority.

101. (New) The method of claim 99, wherein the step of generating the digital signature is performed by applying a signature key to the hash of the software application.

102. (New) The method of claim 101, wherein the mobile device verifies the authenticity of the digital signature by performing the additional steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash;

determining whether the digital signature is authentic by comparing the calculated hash with the recovered hash; and

denying the software application access to the sensitive API where the digital signature is not authenticated.

103. (New) A method of restricting access to application programming interfaces on a mobile device, comprising the steps of:

loading a software application having a digital signature and a signature identification on the mobile device that requires access to one or more application programming interfaces (APIs) having at least one signature identifier;

authenticating the digital signature where the signature identification corresponds with the signature identifier; and

denying the software application access to the one or more APIs where the software application does not include an authentic digital signature .

104. (New) The method of claim 103, wherein the digital signature and signature identification are associated with a type of mobile device.

105. (New) The method of claim 103, comprising the additional step of:

purging the software application from the mobile device where the software application does not include an authentic digital signature. .

106. (New) The method of claim 103, wherein:

the software application includes a plurality of digital signatures and signature identifications; and

the plurality of digital signatures and signature identifications includes digital signatures and signature identifications respectively associated with different types of mobile devices.

107. (New) The method of claim 106, wherein each of the plurality of digital signatures and associated signature identifications are generated by a respective corresponding code signing authority.

108. (New) The method of claim 103, wherein the step of determining whether the software application includes an authentic digital signature comprises the additional steps of:

verifying the authenticity of the digital signature where the signature identification corresponds with respective ones of the at least one signature identifier.

109. (New) The method of claim 107, wherein each of the plurality of digital signatures and signature identifications are generated by its corresponding code signing authority by applying a respective private signature key associated with the code signing authority to a hash of the software application.

110. (New) The method of claim 103, wherein the step of authenticating the digital signature where the signature identification corresponds with the signature identifier comprises the steps of:

verifying that the signature identification corresponds with the signature identifier authenticating the digital signature where signature identification corresponds with the signature identifier comprising the steps of:

storing a public signature key on a mobile device that corresponds to the private signature key associated with the code signing authority which generates the digital signature;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

111. (New) The method of claim 103, wherein:

the mobile device includes a plurality of APIs;

at least one of the plurality of APIs is classified as sensitive;

access to any of the plurality of APIs requires an authentic global signature;

access to each of the plurality of sensitive APIs requires an authentic global signature and an authentic digital signature associated with a signature identification;

the step of determining whether the software application includes an authentic digital signature and signature identification comprises the steps of:

determining whether the one or more APIs to which the software application requires access includes a sensitive API;

determining whether the software application includes an authentic global signature; and

determining whether the software application includes an authentic digital signature and signature identification where the one or more APIs to which the software application requires access includes a sensitive API and the software application includes an authentic global signature; and

the step of denying the software application access to the one or more APIs comprises the steps of:

denying the software application access to the one or more APIs where the software application does not include an authentic global signature; and

denying the software application access to the sensitive API where the one or more APIs to which the software application requires access includes a sensitive API, the software application includes an authentic global signature, and the software application does not include an authentic digital signature and signature identifier required to access the sensitive API.

112. (New) A code signing system for controlling access to application programming interfaces (APIs) having signature identificaters by software applications, the code signing system comprising:

a verification system for authenticating digital signatures provided by the respective software applications to access the APIs where the signature identifications correspond with the signature identificaters of the respective APIs and where a digital signature for a software application is generated with a signature identification corresponding to a signature identificater to access at least one API; and

a control system for allowing access to at least one of the APIs where the digital signature provided by the software application is authenticated by the verification system.

113. (New) The code signing system of claim 112, wherein a virtual machine comprises the verification system and the control system.

114. (New) The code signing system of claim 113, wherein the virtual machine is a Java virtual machine installed on a mobile device.

115. (New) The code signing system of claim 112, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

116. (New) The code signing system of claim 112, wherein the code signing system is installed on a mobile device and the software application is a Java application for a mobile device.

117. (New) The code signing system of claim 112, wherein the digital signature and the signature identification of the software application are generated by a code signing authority.

118. (New) The code signing system of claim 112, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

119. (New) The code signing system of claim 112, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and the verification system uses a public signature key to authenticate the digital signature.

120. (New) The code signing system of claim 119, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

121. (New) The code signing system of claim 112, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

122. (New) The code signing system of claim 112, wherein the APIs provides access to at least one of one or more core functions of a mobile device, an operating system, and hardware on a mobile device.

123. (New) The code signing system of claim 112, wherein verification of a global digital signature provided by the software application is required for accessing any of the APIs.

124. (New) A method of controlling access to application programming interfaces (APIs) having signature identifiers by software applications, the method comprising:

authenticating digital signatures provided by the respective software applications to access the APIs where the signature identifications correspond with the signature identifiers of the respective APIs and where a digital signature for a software application is generated with a signature identification corresponding to a signature identifier to access at least one API; and

allowing access to at least one of the APIs where the digital signature provided by the software application is authenticated.

125. (New) The method of claim 124, wherein one digital signature and one signature identification are provided by the software application access a library of at least one of the APIs.

126. (New) The method of claim 124, wherein the digital signature and the signature identification of the software application are generated by a code signing authority.

127. (New) The method of claim 124, wherein the APIs access at least one of a cryptographic module that implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

128. (New) The method of claim 124, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and a public signature key is used to authenticate the digital signature.

129. (New) The method of claim 128, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the digital signature is authenticated by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

130. (New) The method of claim 124, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

131. (New) The method of claim 124, wherein the APIs provides access to at least one of one or more core functions of a mobile device, an operating system, and hardware on a mobile device.

132. (New) The method of claim 124, wherein verification of a global digital signature provided by the software application is required for accessing any of the APIs

133. (New) A management system for controlling access by software applications to application programming interfaces (APIs) having at least one signature identifier on a subset of a plurality of mobile devices, the management system comprising:

a code signing authority for providing digital signatures and signature identifications to software applications that require access to at least one of the APIs with a signature identifier on the subset of the plurality of mobile devices, where a digital signature for a software application is generated with a signature identification corresponding to a signature identifier, and the signature identifications provided to the software applications comprise those signature identifications that correspond to the signature identifiers that are substantially only on the subset of the plurality of mobile devices; wherein each mobile device of the subset of the plurality of mobile devices comprises

a verification system for authenticating digital signatures provided by the respective software applications to access respective APIs where the digital identifications correspond to the digital identifiers of the respective APIs; and

a control system for allowing the respective software applications to access at least one of the APIs where the digital signatures provided by the respective software applications are authenticated by the verification system.

134. (New) The management system of claim 133, wherein a virtual machine comprises the verification system and the control system.

135. (New) The management system of claim 134, wherein the virtual machine is a Java virtual machine and the software applications are Java applications.

136. (New) The management system of claim 133, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

137. (New) The management system of claim 133, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

138. (New) The management system of claim 133, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and the verification system uses a public signature key to authenticate the digital signature.

139. (New) The management system of claim 138, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

140. (New) The management system of claim 133, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

141. (New) The management system of claim 133, wherein the subset of the plurality of mobile devices comprises mobile devices under the control of at least one of a corporation and a carrier.

142. (New) The management system of claim 133, wherein a global digital signature provided by the software application has to be authenticated before the software application is allowed access to any of the APIs on a mobile device of the subset of the plurality of mobile devices.

143. (New) A method of controlling access by software applications to application programming interfaces (APIs) having at least one signature identifier on a subset of a plurality of mobile devices, the method comprising:

generating digital signatures for software applications with signature identifications corresponding to respective signature identifiers of the APIs; and

providing the digital signatures and the signature identifications to software applications that require access to at least one of the APIs on the subset of the plurality of mobile devices, where the signature identifications provided to the software applications comprise those signature identifications that correspond to the signature identifiers that are substantially only on the subset of the plurality of mobile devices; wherein each mobile device of the subset of the plurality of mobile devices comprises

a verification system for authenticating digital signatures provided by the respective software applications to access respective APIs where the digital identifications correspond to the digital identifiers of the respective APIs; and

a control system for allowing the software application to access at least one of the APIs where the digital signature provided by the software application is authenticated by the verification system.

144. (New) The method of claim 143, wherein a virtual machine comprises the verification system and the control system.

145. (New) The method of claim 144, wherein the virtual machine is a Java virtual machine and the software applications are Java applications.

146. (New) The method of claim 143, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

147. (New) The method of claim 143, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

148. (New) The method of claim 143, wherein at least one of the digital signatures is generated using a private signature key under a signature scheme associated with a signature identification, and the verification system uses a public signature keys to authenticate said at least one of the digital signatures.

149. (New) The method of claim 148, wherein:

at least one of the digital signatures is generated by applying the private signature key to a hash of a software application under the signature scheme; and

the verification system authenticates said at least one of the digital signatures by generating a hash of the software application to obtain a generated hash, applying the public signature key to said at least one of the digital signatures to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

150. (New) The method of claim 143, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

151. (New) The method of claim 143, wherein the subset of the plurality of mobile devices comprises mobile devices under the control of at least one of a corporation and a carrier.

152. (New) A mobile device for a subset of a plurality of mobile devices, the mobile device comprising:

an application platform having application programming interfaces (APIs);

a verification system for authenticating digital signatures and signature identifications provided by the respective software applications to access the APIs; and

a control system for allowing a software application to access at least one of the APIs where a digital signature provided by the software application is authenticated by the verification system;

wherein a code signing authority provides digital signatures and signature identifications to software applications that require access to at least one of the APIs such that the digital signature for the software application is generated according to a signature scheme of a signature identification, and wherein the signature identifications provided to the software applications comprise those signature identifications that are substantially only authorized to allow access on the subset of the plurality of mobile devices.

153. (New) The mobile device of claim 152, wherein a virtual machine comprises the verification system and the control system.

154. (New) The mobile device of claim 153, wherein the virtual machine is a Java virtual machine and the software application is a Java application.

155. (New) The mobile device of claim 152, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

156. (New) The mobile device of claim 152, wherein the APIs of the application platform access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

157. (New) The mobile device of claim 152, wherein the digital signature is generated using a private signature key under the signature scheme, and the verification system uses a public signature key to authenticate the digital signature.

158. (New) The mobile device of claim 157, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

159. (New) The mobile device of claim 152, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

160. (New) A method of controlling access to application programming interfaces (APIs) of an application platform of a mobile device for a subset of a plurality of mobile devices, the method comprising:

receiving digital signatures and signature identifications from software applications that require to access the APIs

authenticating the digital signatures and the signature identifications; and

allowing a software application to access at least one of the APIs where a digital signature provided by the software application is authenticated;

wherein a code signing authority provides the digital signatures and the signature identifications to the software applications that require access to at least one of the APIs such that the digital signature for the software application is generated according to a signature scheme of a signature identification, and wherein the signature identifications provided to the software applications comprise those signature identifications that are substantially only authorized to allow access on the subset of the plurality of mobile devices.

161. (New) The method of claim 160, wherein one digital signature and one signature identification is required for accessing each library of at least one of the APIs.

162. (New) The method of claim 160, wherein the APIs of the application platform access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

163. (New) The method of claim 160, wherein the digital signature is generated using a private signature key under the signature scheme, and a public signature key is used to authenticate the digital signature.

164. (New) The method of claim 163, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the digital signature is authenticated by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

165. (New) The method of claim 160, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

Respectfully submitted,

John V. Biernacki
Reg. No. 40,511
Jones, Day
North Point
901 Lakeside Avenue
Cleveland, OH 44114-1190

Electronic Acknowledgement Receipt	
EFS ID:	1740440
Application Number:	10381219
International Application Number:	
Confirmation Number:	9761
Title of Invention:	Software code signing system and method
First Named Inventor/Applicant Name:	David P Yach
Correspondence Address:	David B Cochran Jones Day North Point 901 Lakeside Avenue Cleveland OH 44114-1190 US - -
Filer:	Stephen D. Scanlon
Filer Authorized By:	
Attorney Docket Number:	555255012423
Receipt Date:	03-MAY-2007
Filing Date:	20-MAR-2003
Time Stamp:	12:14:53
Application Type:	U.S. National Stage under 35 USC 371

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)	Multi Part /.zip	Pages (if appl.)
1	Preliminary Amendment	10289USPCTPrelim.pdf	729990	no	21
Warnings:					
Information:					
		Total Files Size (in bytes):	729990		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/381,219	03/20/2003	David P Yach	555255012423	9761

7590 04/03/2007

David B Cochran
Jones Day
North Point
901 Lakeside Avenue
Cleveland, OH 44114-1190

EXAMINER	
AVERY, JEREMIAH L	
ART UNIT	PAPER NUMBER
2131	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	04/03/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Notice of Non-Compliant Amendment (37 CFR 1.121)	Application No.	Applicant(s)	
	10/381,219	YACH ET AL.	
	Examiner	Art Unit	
	Jeremiah Avery	2131	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

The amendment document filed on *20 March 2003* is considered non-compliant because it has failed to meet the requirements of 37 CFR 1.121 or 1.4. In order for the amendment document to be compliant, correction of the following item(s) is required.

THE FOLLOWING MARKED (X) ITEM(S) CAUSE THE AMENDMENT DOCUMENT TO BE NON-COMPLIANT:

- ☐ 1. Amendments to the specification:
  - ☐ A. Amended paragraph(s) do not include markings.
  - ☐ B. New paragraph(s) should not be underlined.
  - ☐ C. Other _____.
- ☐ 2. Abstract:
  - ☐ A. Not presented on a separate sheet. 37 CFR 1.72.
  - ☐ B. Other _____.
- ☐ 3. Amendments to the drawings:
  - ☐ A. The drawings are not properly identified in the top margin as "Replacement Sheet," "New Sheet," or "Annotated Sheet" as required by 37 CFR 1.121(d).
  - ☐ B. The practice of submitting proposed drawing correction has been eliminated. Replacement drawings showing amended figures, without markings, in compliance with 37 CFR 1.84 are required.
  - ☐ C. Other _____.
- ☒ 4. Amendments to the claims:
  - ☐ A. A complete listing of all of the claims is not present.
  - ☐ B. The listing of claims does not include the text of all pending claims (including withdrawn claims)
  - ☐ C. Each claim has not been provided with the proper status identifier, and as such, the individual status of each claim cannot be identified. Note: the status of every claim must be indicated after its claim number by using one of the following status identifiers: (Original), (Currently amended), (Canceled), (Previously presented), (New), (Not entered), (Withdrawn) and (Withdrawn-currently amended).
  - ☐ D. The claims of this amendment paper have not been presented in ascending numerical order.
  - ☒ E. Other: *See Continuation Sheet*.
- ☐ 5. Other (e.g., the amendment is unsigned or not signed in accordance with 37 CFR 1.4):

_____

For further explanation of the amendment format required by 37 CFR 1.121, see MPEP § 714.

TIME PERIODS FOR FILING A REPLY TO THIS NOTICE:

1. Applicant is given **no new time period** if the non-compliant amendment is an after-final amendment or an amendment filed after allowance. If applicant wishes to resubmit the non-compliant after-final amendment with corrections, the **entire corrected amendment** must be resubmitted.
2. Applicant is given **one month**, or thirty (30) days, whichever is longer, from the mail date of this notice to supply the correction, if the non-compliant amendment is one of the following: a preliminary amendment, a non-final amendment (including a submission for a request for continued examination (RCE) under 37 CFR 1.114), a supplemental amendment filed within a suspension period under 37 CFR 1.103(a) or (c), and an amendment filed in response to a *Quayle* action. If any of above boxes 1. to 4. are checked, the correction required is only the **corrected section** of the non-compliant amendment in compliance with 37 CFR 1.121.

   Extensions of time are available under 37 CFR 1.136(a) only if the non-compliant amendment is a non-final amendment or an amendment filed in response to a *Quayle* action.

   Failure to timely respond to this notice will result in:
   Abandonment of the application if the non-compliant amendment is a non-final amendment or an amendment filed in response to a *Quayle* action; or
   Non-entry of the amendment if the non-compliant amendment is a preliminary amendment or supplemental amendment.

_____ _____

Legal Instruments Examiner (LIE), if applicable Telephone No.

U.S. Patent and Trademark Office Part of Paper No. 20070327

Continuation of 4(e) Other: The numbering of the claims within the preliminary amendment is improper. Claims 1-56 were cancelled and then claims 1-109 were added. However, MPEP 714 states, inter alia, that "The original numbering of the claims must be preserved throughout the prosecution. When claims are canceled, the remaining claims must not be renumbered. For example, when applicant cancels all of the claims in the original specification and adds a new set of claims, the claim listing must include all of the canceled claims with the status identifier (canceled) (the canceled claims may be aggregated into one statement). The new claims must be numbered consecutively beginning with the number next following the highest numbered claim previously presented (whether entered or not) in compliance with 37 CFR 1.126." Thus, the new set of claims cannot begin with claim 1, but must start with claim 57 and ascend in proper numerical order.

AYAZ SHEIKH
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

IFW

PATENT 2131

Attorney Docket No. 555255012423

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	David P. Yach, et al.
Serial No.:	10/381,219
Filed:	March 20, 2003
For:	SOFTWARE CODE SIGNING SYSTEM AND METHOD
Art Unit:	2131
Examiner:	Avery, Jeremiah L.

Commissioner For Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In accordance with the duty of disclosure imposed by 37 C.F.R. § 1.56, applicants hereby advise the United States Patent and Trademark Office of certain references which may be material to the determination of patentability of the above-identified application. The references are identified on the attached Form PTO-1449 and copies of the references are enclosed, if required. Applicants respectfully request that these references be considered and made of record in the present application by completing and returning the enclosed Form PTO-1449.

No fee is believed to be due for entry of this Information Disclosure Statement. However, if any fee should be required, please charge such fee to Jones Day's Deposit Account No. 501432, Reference No. 555255-012423.

Respectfully submitted,

David B. Cochran

David B. Cochran
Reg. No. 39,142
JONES DAY
North Point
901 Lakeside Avenue
Cleveland, Ohio 44114
(216) 586-3939

I hereby certify that this correspondence is being deposited today with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

on Feb 1, 2007

By: Jacquie Owen

CLI-1146631v1

PTO/SB/08B (08-03)
Approved for use through 07/31/2006. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Substitute for form 1449/PTO

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(Use as many sheets as necessary)

Sheet 1 of 1

Complete if Known	
Application Number	10/381,219
Filing Date	March 20, 2003
First Named Inventor	David P. Yach, et al
Art Unit	2131
Examiner Name	Avery, Jeremiah L.
Attorney Docket Number	555255-012423

NON PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T[2]
		Communication of Notices of Opposition (R. 57(1) EPC) dated 26-09-2006 and Working Translation, 16 pages	
		ISO/IEC FCD 7816-9 "Identification cards ...", Part 9: Additional interindustry commands and security attributes", 17.06.1999, S. 8 bis 13, 29 bis 31 (D5), 12 pages	
		ISO/IEC FDIS 7816-8 "Identification cards ...", Part 8: Security related interindustry commands", 25.06.1998, S. 2, 3, 6 bis 13 (D6), 13 pages	
		ISO/IEC 7816-4 "Information Technology - Identification Cards...", Part 4: Interindustry Commands for Interchange", 1995, S. 12 bis 16 (D7), 6 pages	
		Handbuch der Chipkarten, W. Rankl/W. Effing, 3. Auflage Hanser-Verlag Munchen, 1999, S. 197 bis 203, 261 bis 272, 740, 795 bis 797 (D8), 18 pages	

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

1 Applicant's unique citation designation number (optional). 2 Applicant is to place a check mark here if English language Translation is attached.

10391214

Claim Final	Claim Original	Date
	1	
	2	
	3	
	4	
	5	
	6	
	7	
	8	
	9	
	10	
	11	
	12	
	13	
	14	
	15	
	16	
	17	
	18	
	19	
	20	
	21	
	22	
	23	
	24	
	25	
	26	
	27	
	28	
	29	
	30	
	31	
	32	
	33	
	34	
	35	
	36	
	37	
	38	
	39	
	40	
	41	
	42	
	43	
	44	
	45	
	46	
	47	
	48	
	49	
	50	

Claim Final	Claim Original	Date
	51	
	52	
	53	
	54	
	55	
	56	
	57	
	58	
	59	
	60	
	61	
	62	
	63	
	64	
	65	
	66	
	67	
	68	
	69	
	70	
	71	
	72	
	73	
	74	
	75	
	76	
	77	
	78	
	79	
	80	
	81	
	82	
	83	
	84	
	85	
	86	
	87	
	88	
	89	
	90	
	91	
	92	
	93	
	94	
	95	
	96	
	97	
	98	
	99	
	100	

Claim Final	Claim Original	Date
	101	
	102	
	103	
	104	
	105	
	106	
	107	
	108	
	109	
	110	
	111	
	112	
	113	
	114	
	115	
	116	
	117	
	118	
	119	
	120	
	121	
	122	
	123	
	124	
	125	
	126	
	127	
	128	
	129	
	130	
	131	
	132	
	133	
	134	
	135	
	136	
	137	
	138	
	139	
	140	
	141	
	142	
	143	
	144	
	145	
	146	
	147	
	148	
	149	
	150	

If more than 150 claims or 10 actions staple additional sheet here

(LEFT INSIDE)

PATENT APPLICATION FEE DETERMINATION RECORD
Effective January 1, 2003

Application or Docket Number: **10/381214**

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS		
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	13 minus 20=	*
INDEPENDENT CLAIMS	3 minus 3 =	*
MULTIPLE DEPENDENT CLAIM PRESENT		☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐		OR	OTHER THAN SMALL ENTITY	
RATE	FEE		RATE	FEE
BASIC FEE	375.00	OR	BASIC FEE	~~[illegible]~~00 480
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL		OR	TOTAL	900

CLAIMS AS AMENDED - PART II

AMENDMENT A

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	* 13	Minus	** 20	=
Independent	* 2	Minus	*** 3	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
RATE	ADDI-TIONAL FEE		RATE	ADDI-TIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDI-TIONAL FEE		RATE	ADDI-TIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT FEE		OR	TOTAL ADDIT FEE	

MENDMENT C

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	

RATE	ADDI-TIONAL FEE		RATE	ADDI-TIONAL FEE
XS 9=		OR	XS18=	

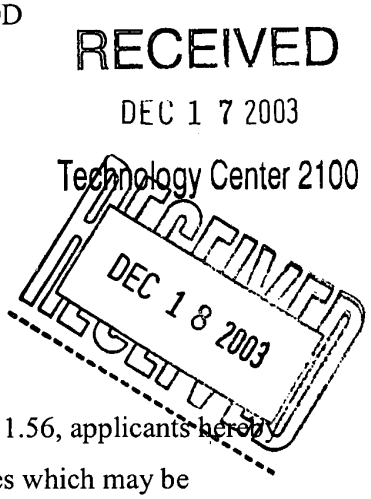
2131

PATENT

Attorney Docket No. 555255012423

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	David P. Yach, et al.
Serial No.:	10/381,219
Filed:	March 20, 2003
For:	CODE SIGNING SYSTEM AND METHOD
Art Unit:	2131
Examiner:	Not yet assigned



Commissioner For Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In accordance with the duty of disclosure imposed by 37 C.F.R. § 1.56, applicants hereby advise the United States Patent and Trademark Office of certain references which may be material to the determination of patentability of the above-identified application. The references are identified on the attached Form PTO-1449 and copies of the references are enclosed. Applicants respectfully request that these references be considered and made of record in the present application by completing and returning the enclosed Form PTO-1449.

No fee is believed to be due for entry of this Information Disclosure Statement. However, if any fee should be required, please charge such fee to Jones Day's Deposit Account No. 501432, Reference No. 555255012423.

Respectfully submitted,

I hereby certify that this correspondence is being deposited today with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

on Dec. 11, 2003

By: A L Pejsar

David B. Cochran
Reg. No. 39,142
JONES DAY
North Point
901 Lakeside Avenue
Cleveland, Ohio 44114
(216) 586-3939

CLI-1146631v1

FORM PTO-1449 (Modified)
U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE

INFORMATION DISCLOSURE STATEMENT BY APPLICANT
(Use several sheets if necessary)

(37 CFR 1.98(b))

Atty Docket No.:	555255012423
Application No.:	10/381,219
Applicants:	David P. Yach, et al.
Filing Date:	March 20, 2003
Group:	2131

U.S. PATENT (AND PATENT PUBLICATION) DOCUMENTS

Exam. Init.		Document No.							Date MM/DD/YYYY	Name	Class	Subclass	Filing Date	
	AA	5	9	7	8	4	8	4	11/02/1999	Apperson et al.				
	AB	6	1	5	7	7	2	1	12/05/2000	Shear et al.				
	AC													
	AD													
	AE													
	AF													
	AG													
	AH													
	AI													
	AJ													
	AK													
	AL													
	AM													

FOREIGN PATENT OR PUBLISHED FOREIGN PATENT APPLICATION

Exam. Init.		Document Number							Publication Date of the Grant	Country or Patent Office	Class	Subclass	Translation	
													Yes	No
	AN	9	9	0	5	6	0	0	02/04/1999	WO				
	AO	0	9	3	0	7	9	3	07/21/1999	EP				
	AP													
	AQ													
	AR													

OTHER DOCUMENTS (Including Author, Title, Date, Relevant pages, Place of Publication***)**

	AS	
	AT	
	AU	

Examiner	Date Considered

EXAMINER: Initial citation considered. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

CLI-1146628v1

10/381,219
12-15-3

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 12/00 — A2

(11) International Publication Number: **WO 99/05600**

(43) International Publication Date: 4 February 1999 (04.02.99)

(21) International Application Number: PCT/US98/15340

(22) International Filing Date: 24 July 1998 (24.07.98)

(30) Priority Data:
08/901,776 28 July 1997 (28.07.97) US

(71) Applicant: APPLE COMPUTER, INC. [US/US]; Law Dept., M/S: 38-PAT, 1 Infinite Loop, Cupertino, CA 95014 (US).

(72) Inventors: GARST, Blaine; 3307 Bay Court, Belmont, CA 94002 (US). SERLET, Bertrand; 218 Colorado Avenue, Palo Alto, CA 94301 (US).

(74) Agents: HECKER, Gary, A. et al.; Hecker & Harriman, Suite 2300, 1925 Century Park East, Los Angeles, CA 90067 (US).

(81) Designated States: CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD AND APPARATUS FOR ENFORCING SOFTWARE LICENSES

(57) Abstract

The present invention comprises a method and apparatus for enforcing software licenses for resource libraries such as an application program interface (API), a toolkit, a framework, a runtime library, a dynamic link library (DLL), an applet (e.g. a Java or ActiveX applet), or any other reusable resource. The present invention allows the resource library to be selectively used only by authorized end user software programs. The present invention can be used to enforce a "per-program" licensing scheme for a resource library whereby the resource library is licensed only for use with particular software programs. In one embodiment, a license text string and a corresponding license key are embedded in a program that has been licensed to use a resource library. The license text string and the license key are supplied, for example, by a resource library vendor to a program developer who wants to use the resource library with an end user program being developed. The license text string includes information about the terms of the license under which the end user program is allowed to use the resource library. The license key is used to authenticate the license text string. The resource library in turn is provided with means for reading the license text string and the license key, and for determining, using the license key, whether the license text string is authentic and whether the license text string has been altered. Resource library functions are made available only to a program having an authentic and unaltered license text string.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

METHOD AND APPARATUS FOR ENFORCING SOFTWARE LICENSES

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION

The present invention relates generally to the distribution of computer software, and more particularly to a method and apparatus for automated enforcement of computer software licenses.

2. BACKGROUND ART

Some computer software programs use so-called "resource libraries" to provide part of their functionality. There is usually a license fee required to use a resource library. Under current schemes, it is not always possible to charge the license fee to all users of a resource library. This problem can be understood by comparing software structures that use resource libraries with basic software structures that do not.

Basic Software Structure

Figure 1 illustrates a basic software structure. In the example of Figure 1, the software comprises two layers. These layers are the operating system 110, and the application program 120. Operating system 110 is responsible for controlling the allocation and usage of hardware resources such as memory, central processing unit (CPU) time, disk space, and peripheral devices. Operating system 110 provides a variety of specific functions that can be

utilized by a variety of software programs such as application program 120. Application program 120 provides specific end user functions, such as word processing, database management, and others. Application program 120 communicates with the computer hardware via functions provided by operating system 110. Operating system 110 provides an interface between hardware 100 and application program 120.

Resource Libraries

Figure 2 illustrates a second software structure. The software structure of Figure 2 contains an additional layer of software, resource library 215, interposed between application program 220 and operating system 110. Resource library 215 provides a pre-packaged set of resources or routines that can be accessed by software programs such as application program 220 during execution. These resources provide higher level functions than those provided by operating system 210. For example, these resources may provide routines for managing a graphical user interface, for communicating with other computers via a network, or for passing messages between program objects. Typically, resource library 215 provides one or more resources or functions that can be used by many different software programs. By using the pre-packaged resources provided by resource library 215, a software program such as application program 220 can be made smaller and program development time can be shortened because the program itself need not include code to provide the functions provided by resource library 215.

In addition to application programs, resource libraries are used by other types of software programs, including device drivers, utility programs and other resource libraries.

Resource library 215 constitutes any set of one or more resources that exists separately from an application program or other software program and that can be used by more than one software program. For example, resource library 215 may comprise an application program interface (API), a toolkit, a framework, a resource library, a dynamic link library (DLL), an applet, or any other reusable resource, including an application program that can be accessed by another program (e.g. by using object linking and embedding (OLE)). Examples of resource libraries include Windows DLL's (DLL's used with the Microsoft Windows (TM) operating environment), the Apple Macintosh (TM) toolkit, the OpenStep API from NeXT Software, Inc., OLE enabled application programs such as Microsoft Word (TM), Java packages, and ActiveX applets.

A software program typically utilizes a resource provided by a resource library by sending an appropriate message to the resource library and supplying the parameters required for the resource to be executed. Assuming the appropriate parameters have been supplied, the resource executes, and an appropriate response message is returned to the requesting program.

A software program may use resources provided by several different resource libraries, a resource library may be used by several different programs, and a resource library may itself use other resource libraries. Figure 3 illustrates a computer system that includes several programs and several resource libraries. In the example of Figure 3, there are two application

programs 300 and 310, and three resource libraries 320, 330, and 340. Application program 300 uses resources provided by operating system 110 and by resource libraries 320 and 330. Application program 310 uses resources provided by operating system 110 and by resource libraries 330 and 340. The resources of resource library 330 are thus shared by application programs 300 and 310.

License Fee

Generally, computer software is licensed to an end user for a fee. The end user pays a single purchase price or license fee in exchange for the right to use the end user program on a computer system. Resource libraries are often packaged or "bundled" with an end user program by the maker of the program such that the end user receives a copy of resource libraries required by a program when the end user buys a copy of the program. The price of the resource library is built into the end user program price. The end user program developer, in turn, pays a royalty to the resource library vendor for the right to bundle and resell the resource library.

Since a resource library can be used with multiple end user programs, once the end user receives a copy of the resource library, the end user can use the resource library with any other program that is compatible with the resource library. In this case, the resource library vendor receives no additional revenue when the vendor's resource library is used with additional programs. Accordingly, it would be desirable for a resource library vendor to be able to ensure that an end user can use the resource library only with programs for which a license fee has been paid to the vendor for use of the

resource library. Thus there is a need for a software mechanism for enforcing software license agreements that automatically ensures that a resource library can only be used by programs that have been licensed for use with the resource library by the resource library vendor.

SUMMARY OF THE INVENTION

The present invention comprises a method and apparatus for enforcing software licenses for resource libraries. The term "resource library" as used herein refers to any reusable software resource that is usable by more than one program or other resource library. The term "resource library" includes, but is not limited to, an application program interface (API), a toolkit, a framework, a runtime library, a dynamic link library (DLL), an applet (e.g. a Java or ActiveX applet), an application program whose functionality can be accessed by other programs (e.g. using OLE) or any other reusable resource. The present invention allows the resource library to be selectively used only by authorized end user software programs. The present invention can be used to enforce a "per-program" licensing scheme for a resource library whereby the resource library is licensed only for use with particular software programs, as well as site licenses and other licensing schemes.

In one embodiment, an access authorization indicator such as a license text string and a corresponding license key are embedded in a program that has been licensed to use a resource library. The license text string and the license key are supplied, for example, by a resource library vendor to a program developer who wants to use the resource library with an end user program being developed.

The license text string includes information about the terms of the license under which the end user program is allowed to use the resource library. In one embodiment, the license key is an algorithmic derivation, such as, for example, a digital signature, of the license text string that is used to

authenticate the license text string. The resource library in turn is provided with a checking routine that includes means for reading the license text string and the license key, and for determining, using the license key, whether the license text string is authentic and whether the license text string has been altered. Resource library functions are made available only to a program having an authentic and unaltered license text string.

In one embodiment, the license key constitutes the resource library vendor's digital signature of the license text string. The resource library has a checking routing for verifying the resource library vendor's digital signature. The resource library is unlocked and made available for use with the requesting program only if the license text string is verified as authentic by the resource library. For a given program, only the resource library proprietor can produce a license key for a particular license agreement that will unlock the resource library for that program and that program only. Any modification of the license key or the license agreement text string in the requesting software program is detected by the checking routine, causing the resource library to remain locked. The license text string may also specify an expiration date for the license, in which case the resource library is unlocked only if the expiration date has not yet occurred.

In one embodiment, a per-site enforcement method is provided, in which any software program present at a given user site works with the resource library once the resource library is provided with the proper per-site license key.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates an example of a software structure.

Figure 2 illustrates an example of a software structure including a resource library.

Figure 3 illustrates an example of a software structure including several application programs and resource libraries.

Figure 4 illustrates an embodiment of a computer system that can be used with the present invention.

Figure 5 illustrates a software structure of one embodiment of the present invention.

Figure 6 illustrates a software structure of one embodiment of the present invention.

Figure 7 is a flow chart illustrating the operation of one embodiment of the present invention.

Figure 8 illustrates a software structure of one embodiment of the present invention.

Figure 9 illustrates a software structure of one embodiment of the present invention.

Figure 10 is a flow start illustrating the operation of one embodiment of the present invention.

Figure 11 is a flow start illustrating the operation of one embodiment of the present invention.

Figure 12 is a flow start illustrating the operation of one embodiment of the present invention.

Figure 13 illustrates a software structure of an embodiment of the present invention using the OpenStep API.

Figure 14 illustrates an embodiment of the invention in which the resource library is an applet.

DETAILED DESCRIPTION OF THE INVENTION

A method and apparatus for enforcing software licenses is described. In the following description, numerous specific details are set forth in order to provide a more thorough description of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known features have not been described in detail so as not to obscure the invention.

Computer System

The present invention can be implemented on any of a variety of computer systems, including, without limitation, network computers, special purpose computers, and general purpose computers such as the general purpose computer illustrated in Figure 4. The computer system shown in Figure 4 includes a CPU unit 400 that includes a central processor, main memory, peripheral interfaces, input-output devices, power supply, and associated circuitry and devices; a display device 410 which may be a cathode ray tube display, LCD display, gas-plasma display, or any other computer display; an input device 430, which may include a keyboard, mouse, digitizer, or other input device; non-volatile storage 420, which may include magnetic, re-writable optical, or other mass storage devices; a transportable media drive 425, which may include magnetic, re-writable optical, or other removable, transportable media, and a printer 450. The computer system may also include a network interface 440, which may include a modem, allowing the computer system to communicate with other systems over a communications network such as the Internet. Any of a variety of other configurations of

computer systems may also be used. In one embodiment, the computer system comprises an Intel Pentium (tm) CPU and runs the Microsoft Windows 95 (tm) operating environment. In another embodiment, the computer system comprises a Motorola 680X0 series CPU and runs the NeXTStep operating system.

When a computer system executes the processes and process flows described herein, it is a means for enforcing software licenses.

The invention can be implemented in computer program code in any desired computer programming language.

Licensing Module

Figure 5 is a block diagram illustrating software components of one embodiment of the present invention. As shown in Figure 5, this embodiment, like the prior art embodiment of Figure 2, includes computer hardware 100, operating system 110, application program 220 and resource library 215. However, the present invention adds two additional components: Program licensing module 500 and resource library licensing module 510. These modules are shown in greater detail in Figure 6.

Figure 6 illustrates program licensing module 500 and resource library licensing module 510 in one embodiment of the present invention. As shown in Figure 6, program licensing module 500 contains license text string 600 and license key 610. License text string 600 contains data specifying terms of the software license agreement under which the resource library vendor

has licensed the program containing program licensing module 510 to use the vendor's resource library. For example, license text string 600 may include the following text:

Table 1: Example License Text String

```
"(c) Copyright 1997.  Resource Library Vendor, Inc. Program A is
licensed to use Resource Library D.  No expiration date.  This
license may not be legally copied or transferred to another
program."
```

In the example shown in Table 1, license text string 600 specifies the name of the resource library vendor ("Resource Library Vendor, Inc.), the name of the program licensed to use the resource library ("Program A"), and the name of the resource library that has been licensed ("Resource Library D"). License text string 600 also indicates that the license has "No expiration date."

License key 610 is algorithmically derived from license text string 600. In one embodiment, license key 610 comprises a digital signature of the resource library vendor.

A digital signature is a mechanism that has been developed to help ensure the integrity of electronic messages. A digital signature is used to authenticate an electronic message and to determine whether an electronic message has been altered.

One form of digital signature uses a message digest. A message digest is a value that is generated when an electronic message is passed through a one way encryption process ("digesting process") such as a hashing routine. An ideal digesting process is one for which the probability that two different electronic messages will generate the same message digest is near zero. In this

form of digital signature, both the originator and the recipient need to know which digesting process is being used. The originator generates the electronic message, and generates a message digest by passing the electronic message through the digesting process. The originator digitally signs the resulting message digest, for example by performing an algorithmic operation on the message digest using the originator's private key. Alternatively, instead of generating a message digest and signing the message digest, a sender may sign the message itself.

To verify the authenticity of a digitally signed message, the recipient obtains the electronic message and the digital signature of the sender. The recipient verifies the digital signature using an appropriate verification process. For example, in one embodiment, the recipient verifies the digital signature by performing an algorithmic process on the digital signature using the sender's public key. The verification process verifies that the electronic message was (1) digitally signed by the sender, and (2) that the electronic message content was not changed from the time that it was signed to the time that the digital signature was verified.

In the present embodiment of the invention, the "message" that is digitally signed is license text string 600. The signer is the resource library vendor. The result is license key 610.

License text string 600 and license key 610 are used by resource library licensing module 510 to verify that a requesting program has been licensed to use the resource library. As shown in Figure 6, resource library licensing module 510 includes a license verification module 620. When a program

requests access to the resource library, resource library licensing module 510 reads license text string 600 and license key 610 from the requesting program. In one embodiment, license text string 600 and license key 610 are sent to the resource library by the requesting program along with a request for access to the resource library. In another embodiment, resource library licensing module 510 reads license text string 600 and license key 610 from a constant definition section of the requesting program.

Resource library licensing module 510 uses license key 610 to verify the content of license text string 600 in the same manner as a digital signature is used to verify an electronic message. Using license verification module 620, resource library licensing module 510 verifies that license text string 600 is authentic (i.e. was generated by the resource library vendor) and unaltered. If the verification process is unsuccessful, indicating that the digital signature is not good, resource library licensing module 510 refuses the requesting program's request for access to the resource library. If the verification process is successful, resource library licensing module 510 inspects the license to determine any license limitations included in license text string 600.

The example license text string 600 shown in Table 1 above identifies "Program A" as the program that is licensed to use the resource library, and states that the license has "No expiration date." Resource library licensing module 510 obtains the name of "Program A" from license text string 600, and checks whether the requesting program is Program A. If the requesting program is a program other than Program A, access to the resource library is denied.

Rather than specifying "No expiration date" as in the present example, license text string 600 may specify an expiration date and/or a beginning date for the license. If any such dates are specified in license text string 600, resource library licensing module 510 checks to make sure that the current date falls within the period of validity of the license prior to granting access to the resource library. If the current date is not within the license's period of validity, the requesting program is denied access to the resource library.

Access Procedure

The process used by a resource library to grant or deny access to a requesting program in one embodiment of the invention is illustrated in Figure 7. In one embodiment, this process occurs the first time a program requests access to a resource library. In another embodiment, this process occurs each time the resource library receives a request for access.

As shown in Figure 7, the process begins with a requesting program making a request to use the resource library at step 700. At step 705, the resource library obtains the requesting program's license text and license key. The license text and license key may, for example, be included in the request, or the resource library may read the license text and license key from a constant declaration area of the requesting program, or the resource library may obtain the license text and license key by some other means.

After obtaining the license text and license key, the resource library verifies the authenticity of the license text, using the license key, at step 710. At step 725, a the resource library determines whether the verification is

successful. If the authenticity of the license text is not verified, access to the resource library is denied at step 730.

If the verification of the authenticity of the license text is successful, the resource library checks the license terms included in the license text at step 735. At step 740, the resource library determines whether a limited validity period is specified in the license text. If no validity period is specified, the process continues on to step 755. If a validity period is specified, the resource library checks whether the validity period has expired at step 745. The validity period will have expired either if the current date is before a beginning date specified in the license text or if the current date is after an expiration date specified in the license text. If the validity period has expired, access to the resource library is denied at step 750.

If the validity period has not expired, processing continues to step 755. At step 755, the resource library determines whether the requesting program is the same program as the program specified in the license text. If the requesting program is not the program specified in the license text, access to the resource library is denied at step 760. If the requesting program is the program specified in the license text, the resource library checks whether there are any other license terms contained in the license text at step 765. If there are no other license terms, access to the resource library is granted at step 770. If there are other license terms, the resource library checks whether those terms are satisfied at step 775. If the terms are not satisfied, access to the resource library is denied at step 780. If the terms are satisfied, access to the resource library is granted at step 785.

The invention may be implemented in the Objective-C language. Objective-C is essentially the ANSI C language with object messaging extensions. A full description of the Objective-C language appears in "Object-Oriented Programming and the Objective-C Language," published by Addison-Wesley (ISBN 0-201-63251-9) (1993), and incorporated by reference herein. However, the invention can also be implemented in any other suitable computer programming language.

As described below, the invention can be implemented by embedding appropriate segments of program code in the source code of a program that uses a resource library and in the source code of the resource library itself. The resource library is compiled to produce an executable implementation which can be linked to a compiled and executable version of the program.

Application Program Interface (API)

In one embodiment of the invention, the resource library is an application program interface ("API"). An API has three major functions: it receives requests from an application program to carry out fundamental operations such as receiving user input or displaying output; it converts each request into a form understandable by the particular operating system then in use; and it receives responses and results from the operating system, formats them in a uniform way, and returns them to the application program.

APIs generally are prepared in an executable implementation which is compiled specifically for the underlying operating system. This is necessary because different operating systems provide different calling mechanisms and

communications methods for such primitive operations as reading and writing a mass storage device. For example, an API may provide a "draw(x,y)" function that can be called by an application program to draw a point at coordinates (x,y) on the display device of a computer system. Upon receipt of a draw(x,y) request from an application program, the API converts the request into a command or function call specific to the operating system then in use. For example, the API might convert the draw(x,y) request into a series of machine instructions to load registers with the x,y values and call an operating system function or generate an interrupt. The person writing the application program need not worry about such details.

In some cases the API refers to or calls functions located in an external function library such as a set of device drivers rather than directly calling the operating system. Device drivers are small executable programs that enable the operating system to address and work with particular hardware devices such as video adapters and printers. Device drivers also constitute a form of resource library.

Depending on the operating system, the API can be prepared in any of several executable formats such as a runtime library, device linked library (DLL), or other executable file. The API is provided to the end user in one of these object code versions, or "implementations," of the API. In industry usage the term API can refer to a definition or specification of functions in the API, to the source code of the API that implements such functions, or to the executable version of such source code which is ultimately distributed to and used by end users. Examples of APIs are the OpenStep API, available from

NeXT Software, Inc., Redwood City, California, and the Visual Basic DLL available from Microsoft Corporation, Redmond, Washington.

The term API as used herein also includes the Java programming language. Rather than being distributed in executable form, Java programs are distributed as packages of "bytecodes." The bytecodes are compiled at runtime into executable code by a Java Virtual Machine (JVM) resident on the computer on which the Java program is run. Different JVM's are used for different computer processors and operating systems. However, all JVM's read the same bytecode. Accordingly, Java bytecode programs and packages are platform independent. Java bytecode programs and packages need only be written in one form. The JVM's take care of adapting the bytecode to different computer platforms. Packages of Java bytecode can be used by different Java programs, and, as such, constitute resource libraries.

Generally the end user can buy the executable version of the API implementation separately from any particular application program from its creator or vendor, or the end user may buy the API implementation bundled with an application program that requires and uses the API to run.

In either case, the API implementation is installed in executable form in the end user's computer system (typically by copying it to a mass storage device such as a hard disk). After the API implementation is installed, the end user can launch (begin running) an application program which uses the API implementation. The application program locates the API implementation on the hard disk and references, calls, or is linked to the API implementation. In operation, when the application program needs to carry

out an operation implemented in the API implementation, such as drawing a line on the screen, the application program calls the appropriate function in the API implementation. The appropriate function in turn tells the operating system (or the device independent windowing extensions, or another device driver) how to execute the desired operation.

A significant advantage of the use of APIs is that an application program, such as a word processor, can be written to communicate only with the API, and not with the operating system. Such an application program can be moved or ported to a different operating system without modifying the program source code. Because of this, application programs written for APIs are said to be operating system independent, meaning that the application program source code can be moved without modification to another computer system having a different operating system, and recompiled and linked with an API implementation prepared for that operating system. The ability to move unmodified application source code to different operating systems is a key advantage of using APIs.

However, from the point of view of API vendors, APIs also have the significant disadvantage that an end user needs only one copy of the API to run multiple application programs which are compatible with the API. Since the API provides generic input, output, and processing functions, it will work with a variety of different end user application programs. Some software vendors desire to restrict use of their API implementations to one application, or to require the end user to purchase a key to the API for each application acquired by the end user, so that the end user pays a different or larger fee to use additional application programs.

The present invention provides a way to arrange a resource library such as an API to work only with particular authorized application or other end user programs.

API License Embodiment

As is well known in the art, the source code of a computer program can be divided into several components including a variables declaration area, a constant declaration area, and a procedure definition area. Figure 9 illustrates an embodiment of the present invention that is used with an API. As shown in Figure 9, in this embodiment, an application program 900 is provided with a LicenseKeyString constant 902 and a LicenseAgreementString constant 904 in the constant declarations area 901 of the application program's source code. In the embodiment of Figure 9, LicenseKeyString 902 and LicenseAgreementString 904 are declared as global string constants.

In one embodiment, LicenseAgreementString 904 contains a text string, prepared by the vendor of the API, that describes in human readable text the license restrictions concerning use of the API applicable to the application program. For example, the LicenseAgreementString may read, "This API is licensed for individual internal use only for concurrent use only with Word Processor application program." The specific text of the LicenseAgreementString is prepared by the licensor of the API. The text can be any arbitrary combination of words, symbols, or numbers.

The LicenseKeyString 904 contains a key corresponding to and based upon the LicenseAgreementString 902. For example, the LicenseKeyString can be a digital signature of the LicenseAgreementString prepared by providing the LicenseAgreementString and a private key of the API vendor to a digital signature process. The precise method of generating the LicenseKeyString is not critical, provided that only the licensor of the API can generate a unique LicenseKeyString corresponding to the LicenseAgreementString. The values of the two strings are created by the vendor of the API and are provided to the person or company that is developing the end user application program (for example, the API vendor can send the two string values to the application program developer by e-mail). The application program developer is instructed by the API vendor to place the string declarations in the source code of the developer's end user application program. The two values may be public, so the API vendor or developer need not keep the values secret or hidden from users of the end user application program. The two strings are compiled into the executable form (or, in the case of Java, the bytecode packages) of the application program. This binds the LicenseKeyString and LicenseAgreementString into the executable code (or bytecode) of the application program.

As further shown in Figure 9, API 920 is provided with an UNLOCK function 923 and a CHECK LICENSE function 921 for testing whether the LicenseKeyString matches the LicenseAgreementString. In the embodiment of Figure 9, the CHECK LICENSE function 921 includes sub-function CHECK 922.

API Procedure

Figure 10 is a flow diagram of processing steps of the UNLOCK function 923. The process of Figure 10 may, for example, be carried out at runtime, when both the application program and the API are compiled, linked, and running.

The UNLOCK function is called by the API upon initialization of the API, for example, upon being called by application program 900 or by some other calling function, object, or program (the "calling entity"). Processing begins at step 1002. The UNLOCK function first checks to see whether the API has been provided with a site license that allows the API to be used with any calling entity on the computer in which the API has been installed. In this embodiment, a site license is indicated by adding an appropriate LicenseKeyString and LicenseAgreementString to the API when the API is installed. This process is described in greater detail below. An appropriate LicenseAgreementString may, for example, be "API site license granted. This API may be used with any application program at the site at which it is installed." The corresponding appropriate LicenseKeyString may, for example, be derived by applying the API vendor's private key and a digital signature process to the LicenseAgreementString.

The process of checking for a site license begins at step 1004 where the UNLOCK function locates and extracts (to the extent they have been provided to the API) a LicenseKeyString and a LicenseAgreementString from within the API. Control is then passed to step 1006 where the function tests whether the API is licensed under a site license for unrestricted use with any application

program. The test of step 1006 is accomplished by verifying the authenticity of the LicenseKeyString and LicenseAgreementString extracted from the API, and, if authentic, determining whether the LicenseAgreementString indicates that a site license has been granted.

The authenticity of the LicenseAgreementString and LicenseKeyString is determined by passing the LicenseAgreementString, the LicenseKeyString, and a copy of the API vendor's public key stored in the API implementation to the CHECK process 922. CHECK process 922 uses a digital signature authentication ("DSA") process to verify the authenticity of the LicenseAgreementString.

The DSA process used by CHECK process 922 can be any digital signature authentication process capable of reading an input string and a key purportedly representing the digital signature of the input string, applying an appropriate authentication process, and determining the validity of the input string by testing whether the key constitutes the signatory's digital signature of the input string. An exemplary DSA process is disclosed, for example, in U.S. Patent Application Serial No. 08/484,264, "Method and Apparatus for Digital Signature Authentication," assigned to the assignee hereof. The DSA technology of RSA Data Security, Inc. also can be adapted for use with the invention. A per-session cache can be used to improve execution speed of the CHECK process.

If the LicenseKeyString is determined to be the API vendor's valid digital signature of the LicenseAgreementString, the LicenseAgreementString is inspected to determine whether it indicates that a site license has been

granted. If the LicenseAgreementString does so indicate, the test of step 1006 succeeds and control is passed to step 1014. At this point the UNLOCK function returns a positive result to the calling entity, and allows the calling entity to use the API.

If the test of step 1006 fails, control is passed to step 1008 where the UNLOCK function extracts and reads the LicenseKeyString and LicenseAgreementString from a data segment (for example, the compiled constant declarations area) of the calling entity. Alternatively, the calling entity may transmit the LicenseKeyString and the LicenseAgreementString to the API. Having obtained the calling entity's LicenseKeyString and LicenseAgreementString, control is passed to step 1010 where the function tests whether the calling entity is licensed to use the API. This test comprises two parts. One part, using CHECK process 922 as described above, determines whether the LicenseAgreementString is a LicenseAgreementString validly issued by the API vendor. A second part examines the LicenseAgreementString for the terms of the included license, and determines whether those terms are met. If the result is positive then control is passed to step 1014. At this point, use of the API with the calling entity is authorized and the API returns control to the calling entity so that the calling entity resumes normal execution.

If the result is negative then the calling entity is not licensed to use the API, and control is passed to step 1012. At step 1012 the API generates an error message such as "API Not Licensed For Use With This Application program," and declines access to the calling entity.

Steps 1006 and 1010 carry out the license tests by calling the CHECK LICENSE function 921 shown in Figure 9 and Figure 11. Processing steps of the CHECK LICENSE function 921 are illustrated in Figure 11.

The process flow of the CHECK LICENSE function starts at step 1102. Control is passed to step 1104 where the CHECK LICENSE function assembles the LicenseKeyString 902, LicenseAgreementString 904, and a copy of the API vendor's public key 1106 as function call arguments, in preparation for calling the CHECK function 922. As discussed more fully below, the public key 1106 is prepared by the API vendor based upon a secret private key. The three arguments are passed to the CHECK function at step 1108.

If the CHECK function (described in greater detail below) returns a FAIL or false state, control is passed to step 1124 and the CHECK LICENSE function itself returns a fail state. If the CHECK function returns a PASS or true state, control is passed to step 1112 where the CHECK LICENSE function checks the terms of the license specified in the LicenseAgreementString. At step 1114, the CHECK LICENSE function checks whether the name of the calling entity is the same as the name of the licensed entity specified in the LicenseAgreementString. If the name of the calling entity is incorrect, control passes to step 1124, where a fail message is passed to the UNLOCK function.

If the name of the calling entity is correct, the CHECK LICENSE function tests whether the LicenseAgreementString contains an expiration date at step 1116. An expiration date can be placed in the LicenseAgreementString by the API vendor to establish a termination date after which use of the API by the calling entity is no longer allowed. CHECK

LICENSE may, for example, test for an expiration date by searching for a text string that indicates an expiration date, such as, for example, "expiration date" or "valid until."

If the test of step 1116 is positive, control is passed to step 1118 where the CHECK LICENSE function tests whether the current date, as maintained, for example by a computer clock or operating system, is greater than the expiration date found in the LicenseAgreementString. If the test of step 1118 passes, control is passed to step 1120. If the test of step 1118 fails, then CHECK LICENSE returns a FAIL message at block 1124.

At step 1120, the CHECK LICENSE function checks whether the LicenseAgreementString specifies any additional license terms. If there are no other terms, CHECK LICENSE returns a PASS message at block 1126. If there are other terms, CHECK LICENSE determines whether those terms are met at block 1122. If any of the other terms are not met, CHECK LICENSE returns a FAIL message at block 1124. If all of the additional terms are met, CHECK LICENSE returns a PASS message at block 1126.

The operation of the CHECK function called by CHECK LICENSE at block 1108 is illustrated in Figure 12. As shown in Figure 12, the purpose of the CHECK function is to verify the authenticity of a license agreement string by verifying that a corresponding license key string constitutes a valid digital signature of the license agreement string. The CHECK function begins at step 1202 and receives as input a LicenseKeyString, a LicenseAgreementString, and a vendor's public key in step 1203. The public key is generated by the resource library vendor using any known public/private key pair generation process, as

is well known in the field of cryptography. For example, key generation using Fast Elliptical Encryption (FEE) can be done, or Diffie-Hellman key generation can be used.

In step 1204 the CHECK function verifies that the LicenseKeyString comprises the digital signature of the LicenseAgreementString. In step 1208, the CHECK function tests whether the verification of step 1204 successfully verified the LicenseKeyString as comprising the digital signature of the LicenseAgreementString. If so, the LicenseAgreementString is valid, and CHECK returns a Boolean true or pass value. If not, the LicenseAgreementString is invalid, and CHECK returns false or failure.

Since the LicenseKeyString of the present embodiment comprises the digital signature of the LicenseAgreementString, the LicenseAgreementString cannot be changed in any way without the change being detected. Stated more generally, because the identifier (e.g. the LicenseKeyString) of the invention is a unique key mathematically derived from a particular text string that specifies license terms for a particular end user program (e.g. the LicenseAgreementString), the identifier can be used to detect any changes to the license terms. This prevents unauthorized modification of the text string from extending use of a resource library to an unlicensed program. For example, if an end user attempts to modify the expiration date using a debugger or machine language editor, the identifier will no longer match the license text string. Without knowing the private key of the vendor, the end user cannot generate a matching identifier.

When a 127-bit private key's is used by the vendor to create the identifier used in the present invention, a determined hacker attempting to forge the private key would need to exhaustively search the 127-bit space, requiring extensive computing resources and an impractical amount of time. Thus, the protection provided by the present invention cannot easily be cracked and the security of the invention as a whole is extremely high.

In addition to allowing per program resource library licensing, if the API vendor or licensor desires to grant a site license for the API to the end user, so that the API is licensed for use with any number of application programs, the API may be provided with a LicenseKeyString and a LicenseAgreementString providing for such unrestricted use. In this embodiment, the API vendor provides a site license key string to the end user as authorization to use the API with any number of applications and other end user programs at that site. The site license key string comprises a digital signature of a site license agreement string created by the API vendor. The site license agreement string may be pre-embedded in the API by the vendor. During installation of the API, an installation program provided with the API asks the end user whether a site license key is known. If so, the end user enters the site license key, and the installation program writes the site license key to a reserved location in the API. Thereafter, when the API initializes, the API tests for the presence of the site license key. If it is present, and it comprises a valid digital signature for the site license text string stored elsewhere in the API, the API is permitted to be used with any application program which is calling it.

OpenStep API

In one embodiment of the invention, the API used is the object-oriented OpenStep API 820 shown in Figure 8. A specification of the OpenStep API has been published by NeXT Software, Inc. under the title "OPENSTEP SPECIFICATION," dated October 18, 1994. Implementations of the OpenStep API include implementations for the Windows NT and Solaris operating systems that are available from NeXT Software, Inc. and SunSoft, Inc., respectively.

As shown in Figure 8, the OpenStep API 820 comprises computer program code organized as an Application Kit 802, Foundation Kit 808, and Display Postscript™ system 804. (Display Postscript™ is a trademark of Adobe Systems Incorporated.)

Application Kit 802 provides basic resources for interactive application programs that use windows, draw on the screen, and respond to user actions on the keyboard and mouse. Application Kit 802 contains components that define the user interface. These components include classes, protocols, C language functions, constants and data types that are designed to be used by virtually every application running under the OpenStep API. A principal purpose of Application Kit 802 is to provide a framework for implementing a graphical, event-driven application.

Foundation Kit 808 provides fundamental software functions or building blocks that application programs use to manage data and resources. Foundation Kit 808 defines basic utility classes and facilities for handling

multi-byte character sets, object persistency and distribution, and provides an interface to common operating system facilities. Foundation Kit 808 thus provides a level of operating system independence, enhancing the developer's ability to port an application program from one operating system to another.

Display Postscript system 804 provides a device-independent imaging model for displaying documents on a computer screen. Display Postscript is defined by Adobe Systems Incorporated. Display Postscript system 804 provides an application-independent interface to Postscript.

Separate from the API 820, but also logically located between the application program 800 and the operating system 810, is a set of device dependent windowing extensions 806. Extensions 806 enable Display Postscript system 804 to communicate with specific graphics and display hardware in the end user's computer system, such as the video memory or other video display hardware.

Figure 13 illustrates an embodiment of the invention used with the OpenStep API of Figure 8. As shown in Figure 13, in this embodiment, the license text string and the license key string of the invention are implemented in a property list area 1302 (Info.plist) of the application program code 800. Two string properties are added to the property list area 1302: NSLicenseAgreement 1304, that stores the software license terms applicable to application program 800, and NSLicenseKey 1306, that stores the license key corresponding to NSLicenseAgreement 1304. In this embodiment, as in the embodiment of Figure 9, NSLicenseKey 1306 is derived from the

NSLicenseAgreement string 1304 generated from the license agreement string using a digital signature process and a vendor's private key.

Example values of the two strings placed in the Info.plist are shown in Table 2.

Table 2 -- Info.plist Strings

```
NSLicenseKey = "Ab76LY2GbbO0GqK2KY17BqHy35";

NSLicenseAgreement = "(c) Copyright 1996, EOF AddOnTools
Inc., ReportWriter licensing agreement: This is
demonstration software valid until November 2, 1996.
This software cannot be legally copied.";
```

In the OpenStep embodiment of Figure 13, the UNLOCK function 1308 is implemented as part of Application Kit 802. In one embodiment, UNLOCK function 1308 is implemented by adding appropriate code to a non-redefinable private Application Kit function (such as, for example, _NXAppZone() in NSApplication.m). An example of source code that may be added is shown in Table 3.

Table 3 -- UNLOCK Code added in OpenStep API Implementation

```
static BOOL licenseChecked = NO;
if (! licenseChecked)
     {
     NSDictionary *info;
     NSString *key, *agreement;
     /* First check the unlimited (per-site) license */
     info = [NSDictionary
dictionaryWithContentsOfFile:@"/OpenStep/AppKit.dll/Info
.plist"]; // real path TBD
     key = [info objectForKey:@"NSLicenseKey"];
     agreement = [info
objectForKey:@"NSLicenseAgreement"];
     if (!NSCheckLicense(key, agreement))
          {
          /* now check for the per-app license */
          info = [[NSBundle mainBundle] infoDictionary];
```

```
            key = [info objectForKey:@"NSLicenseKey"];
            agreement = [info
      objectForKey:@"NSLicenseAgreement"];
            if (!NSCheckLicense(key, agreement))
                  {
                  NSLog(@"*** Sorry no valid license for
      %@", [NSApp appName]);
                  }
            }
      licenseChecked = YES;
      }
```

The NSCheckLicense() function, which is called twice in the code segment of Table 3, as shown in Figure 13, is implemented in the Foundation Kit portion 808 of the OpenStep API 820. The NSCheckLicense function 1310 corresponds to the CHECK LICENSE function 921 illustrated in Figure 9. The NSCheckLicense function 1310 verifies NSLicenseAgreement string 1304 using NSLicenseKey string 1306 and a digital signature authentication process. The NSCheckLicense function 1310 has the following definition:

```
extern BOOL NSCheckLicense(NSString *licenseKey,
      NSString *licenseAgreement);
```

The NSCheckLicense function 1310, like the Check License function 921 of Figure 9, applies a CHECK function 1312 to NSLicenseAgreement string 1304 and NSLicenseKey 1306, using the API vendor's public key, to determine the validity of NSLicenseAgreement string 1304. In the embodiment of Figure 13, CHECK function 1312 includes in its code a copy of the API vendor's public key 1314.

In the embodiment of Figure 13, API 820 includes a "GEN" process 1316 that can be used by an API vendor to rapidly generate license key strings for use by CHECK function 1312. GEN process 1316 receives as input a license agreement string and a secret private key, and produces as output a licensing

key string, using a digital signature generating process. The private key may, for example, be a 127-bit private key, although any other size private key may be used. The signature generating process used by GEN process 1316 is compatible with the digital signature authentication process used by CHECK function 1312. GEN process 1316 itself can be made entirely public and implemented in the API provided that the private key of the API vendor is maintained in secrecy. For example, the GEN process can be part of the OpenStep API Foundation Kit 808 as shown in Figure 13. GEN also can be maintained in a separate program module.

The logical relationship between GEN and CHECK is:

CHECK(GEN(LicenseAgreementString, PrivateKey), Public Key, LicenseAgreementString) => YES

CHECK(random1, random2) => NO with a very high probability

In one embodiment of the invention, a shell is provided for the GEN process. The shell can receive as input a license agreement template string, such as:

```
(c) Copyright 1995, %@, %@ licensing agreement; Demo
software valid until %@; This agreement cannot be
legally copied
```

where %@ represents additional data to be provided by the API vendor. The shell then asks the user (i.e. the API vendor) to input the additional data, for example a company name, a product name, an expiration date, from which the shell generates a specific license agreement string. The shell then asks for the private key and applies GEN to create a corresponding license key.

The same shell can be used for per-program license keys or per-site license keys, using different templates.

In one embodiment of the invention, an installer program is provided for installing a resource library on an end user computer. The installer program is provided with a feature enabling the end user to provide a site license key during installation. For example, if the resource library is the OpenStep API, additional code is added to the OpenStep API installer program. The user is asked during the installation of the resource library if the user has obtained a per-site license. If the user replies yes, the user is asked to enter the site license key string. In one embodiment, the user is also asked to enter the site license agreement string. In another embodiment, the site license agreement string is stored in the resource library, such as, for example, in the OpenStep API DLL Application Kit's Info.plist resource file. The site license key and site license agreement are validated by the CHECK LICENSE function as described above. Use of the resource library is permitted only if the site license key string input by the user corresponds to (i.e. is found to comprise the resource library vendor's digital signature of) the site license agreement string.

Java

The present invention may be used with resource libraries such as Java class files, Java applets, and Java bytecode packages. Figure 14 illustrates an embodiment of the invention in which the resource library is a Java applet. In the embodiment shown in Figure 14, an applet is called from an HTML page 1402 via applet tag 1404. Applet tag 1404 includes the name of the

applet's class file and applet parameters 1406. Applet parameters 1406 include a license agreement string parameter 1408 and a license key string parameter 1410. License agreement string parameter 1408 specifies a license agreement string that contains terms of a license to use the called for applet. License key string parameter 1410 specifies a license key used to authenticate the license agreement string. As in other embodiments of the invention, in this embodiment, the license key string comprises a digital signature by the resource library (applet) vendor of the license agreement string. Table 4 illustrates an example of applet tag 1404.

Table 4

```
<APPLET CODE="Applet.class"  WIDTH=250 HEIGHT=75>
<PARAM NAME=LicenseAgreementString VALUE="Web page
orderform.html licensed to use applet 'Applet.class'>
<PARAM NAME=LicenseKeyString VALUE="4kd094kak2rtx0kzq">
</APPLET>
```

In the example of Table 4, the license agreement string specifies the name of the HTML page ("orderform.html") and the name of the licensed applet ("applet.class").

As shown in Figure 14, applet 1434 is accessed when HTML page 1402 is loaded by a HTML browser 1430 running in a client computer 1420. In the embodiment of Figure 14, HTML browser 1430 runs on top of an API 1424 which in turn runs on top of operating system 1422. HTML browser 1430 includes a Java virtual machine 1432 for running Java applets.

Upon encountering applet tag 1404 while loading HTML page 1402, HTML browser 1430 retrieves the class files that constitute applet 1434 from storage locations on client computer 1420 and/or from one or more server

computers, as applicable. One of the class files includes CheckLicense class file 1436. After HTML browser 1430 has retrieved all the required components of applet 1434, applet 1434 is initialized. During initialization, or at a later time, the CheckLicense function provided by CheckLicense class file 1436 is called. As in other embodiments of the invention, the CheckLicense function determines whether the requesting entity (HTML page 1402) possesses a valid license to use the requested resource (applet 1434) by testing the authenticity of the license specified by LicenseAgreementString parameter 1408 using the license key specified by LicenseKeyString parameter 1410 and the applet vendor's public key 1438. If the CheckLicense function determines that HTML page 1402 possesses a valid license, applet 1434 is allowed to execute. If not, execution of applet 1434 is terminated, and an error message is sent to HTML browser 1430.

Thus, an improved method and apparatus for enforcing software licenses has been presented. Although the present invention has been described with respect to certain example embodiments, it will be apparent to those skilled in the art that the present invention is not limited to these specific embodiments. For example, although the invention has been described for use in stand-alone computer systems, the invention can be used to enforce licenses in a network environment as well. Further, although the operation of certain embodiments has been described in detail using specific software programs and certain detailed process steps, different software may be used, and some of the steps may be omitted or other similar steps may be substituted, without departing from the scope of the invention. Other embodiments incorporating the inventive features of the present invention will be apparent to those skilled in the art.

CLAIMS

1. In a computer operating environment comprising a software program and a software resource, an apparatus for limiting use of said software resource comprising:

an access authorization indicator associated with said software program;

means in said software resource for reading said access authorization indicator;

means in said software resource for determining whether said access authorization indicator is valid;

means for allowing access by said software program to said software resource only if said access authorization indicator is determined to be valid.

2. The apparatus of claim 1 wherein said access authorization indicator comprises terms of a license for use of said software resource.

3. The apparatus of claim 1 wherein said access authorization indicator comprises terms of a site license.

4. The apparatus of claim 1 wherein said access authorization indicator is embedded in said software program.

5. The apparatus of claim 1 wherein said software resource comprises an API.

6. The apparatus of claim 1 wherein said software resource comprises a runtime library.

7. The apparatus of claim 1 wherein said software resource comprises a dynamic link library.

8. The apparatus of claim 1 wherein said software resource comprises an applet.

9. The apparatus of claim 1 wherein said software resource comprises a bytecode package.

10. The apparatus of claim 1 wherein said software resource comprises an OLE enabled application program.

11. The apparatus of claim 4 wherein said access authorization indicator is specified in a constant declaration area of said software program.

12. The apparatus of claim 4 wherein said access authorization indicator comprises a property of a property list of said software program.

13. The apparatus of claim 1 further comprising an identifier associated with said access authorization indicator and wherein said means for determining the validity of said access authorization indicator comprises means for determining whether said access authorization indicator is valid based on said identifier.

14. The apparatus of claim 13 further comprising means for receiving said identifier from an end user.

15. The apparatus of claim 14 further comprising means for storing said identifier in said software resource.

16. The apparatus of claim 13 wherein said identifier is embedded in said software program.

17. The apparatus of claim 13 wherein said identifier comprises a digital signature of said access authorization indicator.

18. The apparatus of claim 16 wherein said identifier is specified in a constant declaration area of said software program.

19. The apparatus of claim 16 wherein said identifier comprises a property of a property list of said software program.

20. The apparatus of claim 17 wherein said means for determining whether said access authorization indicator is valid based upon said identifier comprises a means for digital signature authentication.

21. The apparatus of claim 2 further comprising means for determining whether said terms of said license are met.

22. The apparatus of claim 13 wherein:

said software program comprises said access authorization indicator and said identifier;

said access authorization indicator comprises terms of a license for use of said software resource;

said identifier comprises a digital signature of said access authorization indicator.

23. In a computer operating environment, a method for limiting use of a software resource comprising:

receiving a request from a software program to use said resource;

obtaining an access authorization indicator associated with said software program;

determining whether said access authorization indicator is valid;

allowing said software program to use said software resource only if said access authorization indicator is determined to be valid.

24. The method of claim 23 wherein said access authorization indicator comprises terms of a license for use of said software resource.

25. The method of claim 24 wherein said license comprises a site license.

26. The method of claim 23 wherein said access authorization indicator is embedded in said software program.

27. The method of claim 23 wherein said software resource comprises an API.

28. The method of claim 23 wherein said software resource comprises a runtime library.

29. The method of claim 23 wherein said software resource comprises a dynamic link library.

30. The method of claim 23 wherein said software resource comprises an applet.

31. The method of claim 23 wherein said software resource comprises a bytecode package.

32. The method of claim 23 wherein said software resource comprises an OLE enabled application program.

33. The method of claim 26 wherein said access authorization indicator is specified in a constant declaration area of said software program.

34. The method of claim 26 wherein said access authorization indicator comprises a property of a property list area of said software program.

35. The method of claim 23 wherein said determining the validity of said access authorization indicator comprises determining whether said access authorization indicator is valid based on an identifier associated with said access authorization indicator.

36. The method of claim 35 further comprising accepting said identifier from a user.

37. The method of claim 36 further comprising storing said identifier in said software resource.

38. The method of claim 35 wherein said identifier is embedded in said software program.

39. The method of claim 35 wherein said identifier comprises a digital signature of said access authorization indicator.

40. The method of claim 38 wherein said identifier is specified in a constant declaration area of said software program.

41. The method of claim 38 wherein said identifier comprises a property of a property list area of said software program.

42. The method of claim 35 wherein a digital signature authentication means is used in determining whether said access authorization indicator is valid based upon said identifier.

43. The method of claim 24 further comprising determining whether said terms of said license are met.

44. The method of claim 35 wherein:

said software program comprises said access authorization indicator and said identifier;

said access authorization indicator comprises terms of a license for use of said software resource;

said identifier comprises a digital signature of said access authorization indicator.

45. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform a method for limiting use of a software resource, said method comprising:

receiving a request from a software program to use said resource;

obtaining an access authorization indicator associated with said software program;

determining whether said access authorization indicator is valid;

allowing said software program to use said software resource only if said access authorization indicator is determined to be valid.

46. The program storage device of claim 45 wherein said access authorization indicator comprises terms of a license for use of said software resource.

47. The program storage device of claim 46 wherein said license comprises a site license.

48. The program storage device of claim 45 wherein said access authorization indicator is embedded in said software program.

49. The program storage device of claim 45 wherein said software resource comprises an API.

50. The program storage device of claim 45 wherein said software resource comprises a runtime library.

51. The program storage device of claim 45 wherein said software resource comprises a dynamic link library.

52. The program storage device of claim 45 wherein said software resource comprises an applet.

53. The program storage device of claim 45 wherein said software resource comprises a bytecode package.

54. The program storage device of claim 45 wherein said software resource comprises an OLE enabled application program.

55. The method of claim 48 wherein said access authorization indicator is specified in a constant declaration area of said software program.

56. The program storage device of claim 48 wherein said access authorization indicator comprises a property of a property list area of said software program.

57. The program storage device of claim 45 wherein said determining the validity of said access authorization indicator comprises determining whether said access authorization indicator is valid based on an identifier associated with said access authorization indicator.

58. The program storage device of claim 57 wherein said method further comprises accepting said identifier from a user.

59. The program storage device of claim 58 wherein said method further comprises storing said identifier in said software resource.

60. The program storage device of claim 57 wherein said identifier is embedded in said software program.

61. The program storage device of claim 57 wherein said identifier comprises a digital signature of said access authorization indicator.

62. The program storage device of claim 60 wherein said identifier is specified in a constant declaration area of said software program.

63. The program storage device of claim 60 wherein said identifier comprises a property of a property list area of said software program.

64. The program storage device of claim 57 wherein a digital signature authentication means is used in determining whether said access authorization indicator is valid based upon said identifier.

65. The program storage device of claim 46 in which said method further comprises determining whether said terms of said license are met.

66. The program storage device of claim 57 wherein:

said software program comprises said access authorization indicator and said identifier;

said access authorization indicator comprises terms of a license for use of said software resource;

said identifier comprises a digital signature of said access authorization indicator.

67. An article of manufacture comprising:

a computer readable medium having computer readable program code embodied therein for accessing a resource library, said computer readable program code in said article of manufacture comprising:

computer readable program code embodying an access authorization indicator for accessing said resource library.

68. The article of manufacture of claim 67 wherein said access authorization indicator comprises terms of a license for use of said software resource.

69. The article of manufacture of claim 67 wherein said computer readable program code comprises a software program and wherein said access authorization indicator is embedded in said software program.

70. The article of manufacture of claim 67 wherein said software resource comprises an API.

71. The article of manufacture of claim 67 wherein said software resource comprises a runtime library.

72. The article of manufacture of claim 67 wherein said software resource comprises a dynamic link library.

73. The article of manufacture of claim 67 wherein said software resource comprises an applet.

74. The article of manufacture of claim 67 wherein said software resource comprises a bytecode package.

75. The article of manufacture of claim 67 wherein said software resource comprises an OLE enabled application program.

76. The article of manufacture of claim 69 wherein said access authorization indicator is specified in a constant declaration area of said software program.

77. The article of manufacture of claim 69 wherein said access authorization indicator comprises a property of a property list of said software program.

78. The article of manufacture of claim 67 further comprising computer readable program code embodying an identifier associated with said access authorization indicator.

79. The article of manufacture of claim 78 wherein said identifier is embedded in said software program.

80. The article of manufacture of claim 78 wherein said identifier comprises a digital signature of said access authorization indicator.

81. The article of manufacture of claim 78 wherein said identifier is specified in a constant declaration area of said software program.

82. The article of manufacture of claim 78 wherein said identifier comprises a property of a property list of said software program.

83. The article of manufacture of claim 78 wherein:

said software program comprises said access authorization indicator and said identifier;

said access authorization indicator comprises terms of a license for use of said software resource;

said identifier comprises a digital signature of said access authorization indicator.

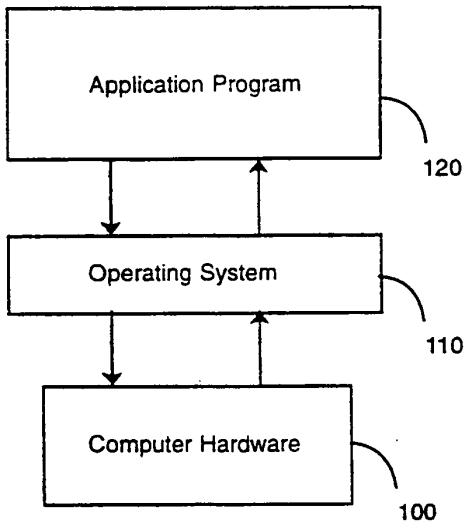

Application Program
120
Operating System
110
Computer Hardware
100

FIG. 1

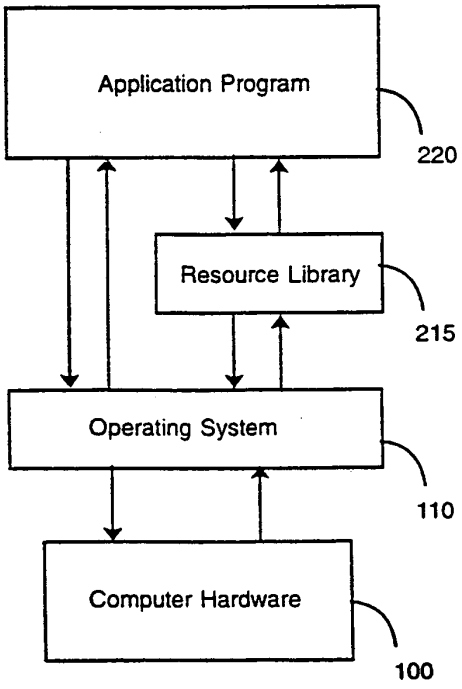

Application Program
220
Resource Library
215
Operating System
110
Computer Hardware
100

FIG. 2

FIG. 3

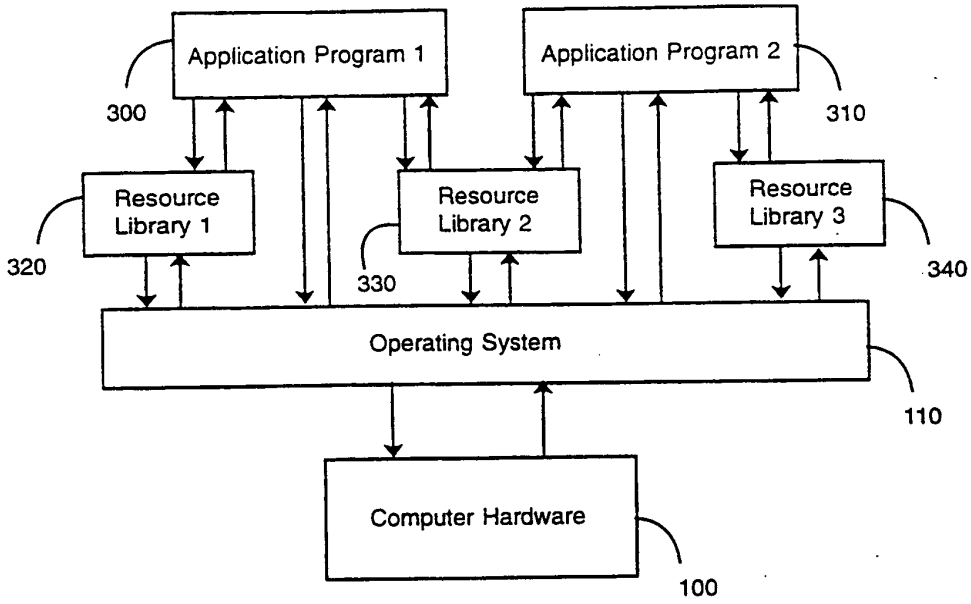
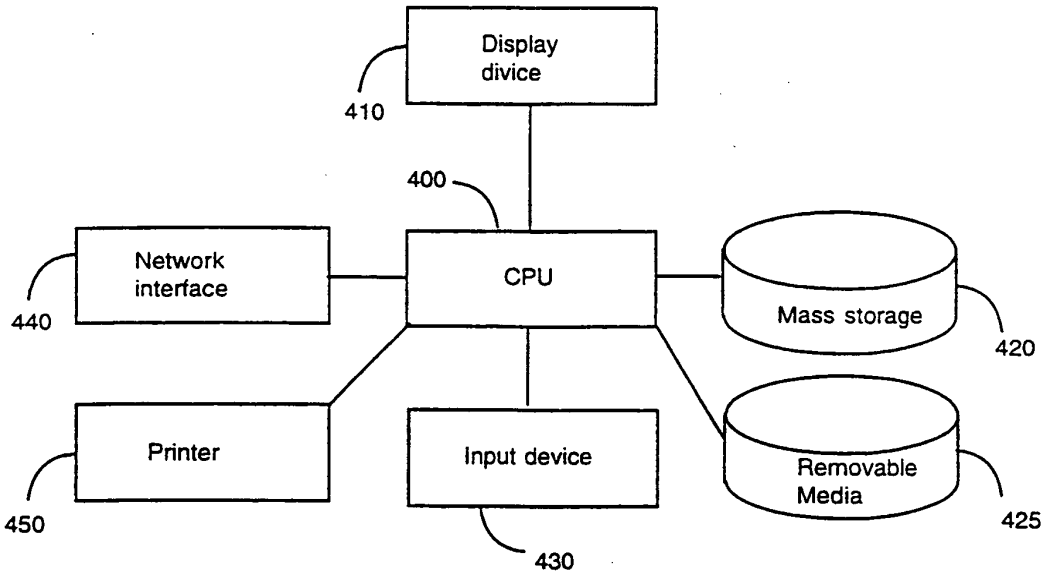

Application Program 1
Application Program 2
300
310
Resource Library 1
Resource Library 2
Resource Library 3
320
330
340
Operating System
110
Computer Hardware
100

FIG. 4


Display divice
410
400
Network interface
440
CPU
Mass storage
420
Printer
450
Input device
430
Removable Media
425

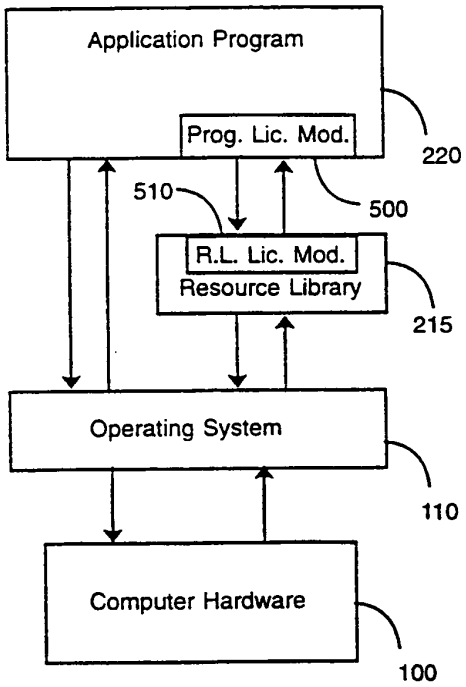

Application Program
Prog. Lic. Mod.
220
510
500
R.L. Lic. Mod.
Resource Library
215
Operating System
110
Computer Hardware
100

FIG. 5

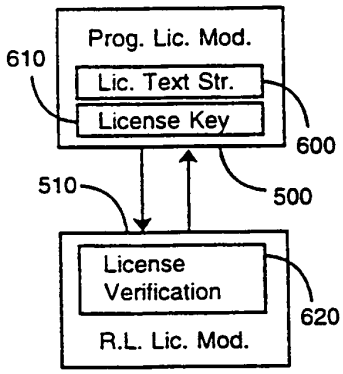

Prog. Lic. Mod.
610
Lic. Text Str.
License Key
600
510
500
License
Verification
620
R.L. Lic. Mod.

FIG. 6

FIG. 7

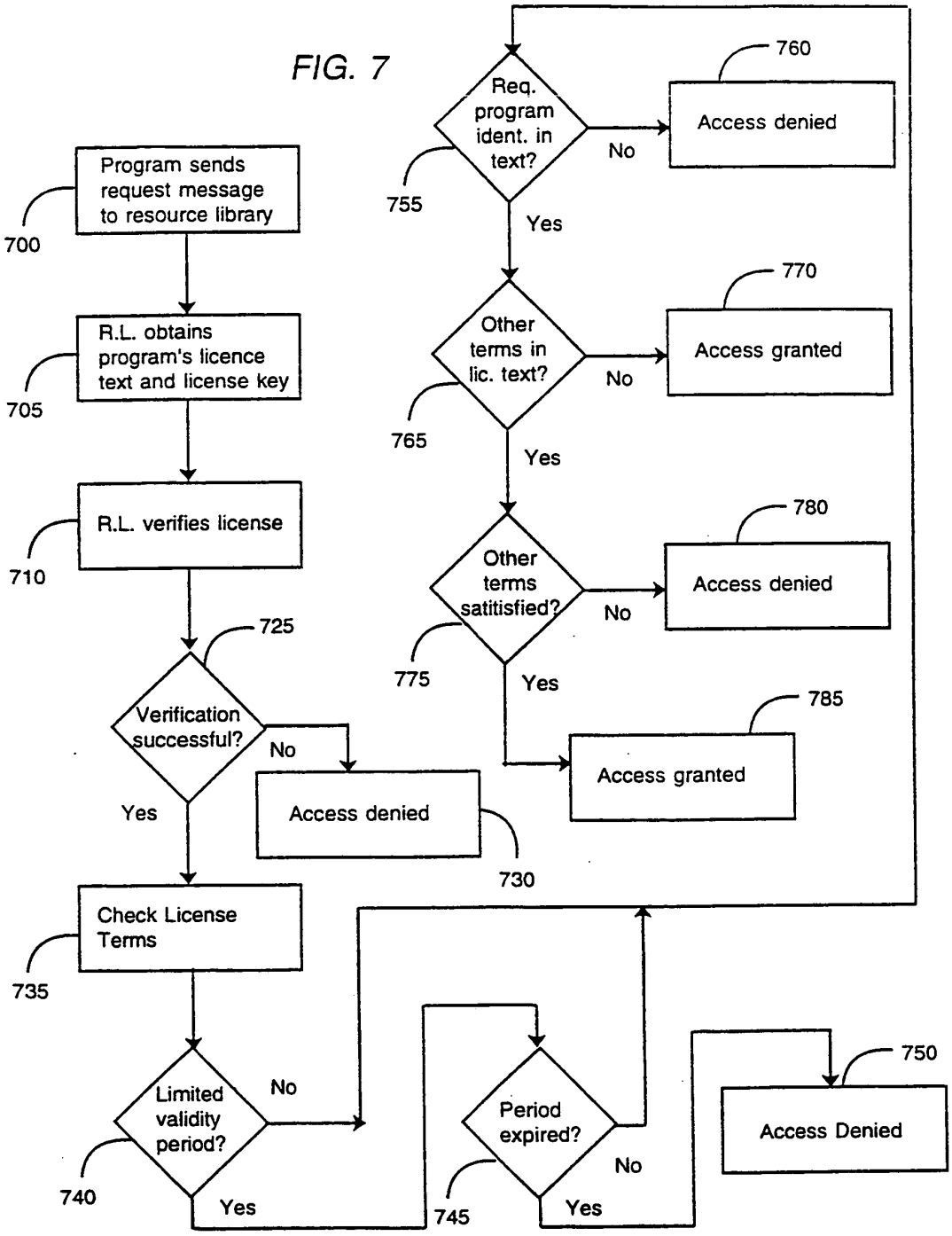

Program sends request message to resource library
700
R.L. obtains program's licence text and license key
705
R.L. verifies license
710
725
Verification successful?
No
Access denied
730
Yes
Check License Terms
735
Limited validity period?
740
No
Yes
Req. program ident. in text?
755
No
760
Access denied
Yes
Other terms in lic. text?
765
No
770
Access granted
Yes
Other terms satitisfied?
775
No
780
Access denied
Yes
785
Access granted
Period expired?
745
No
Yes
750
Access Denied

FIG. 8

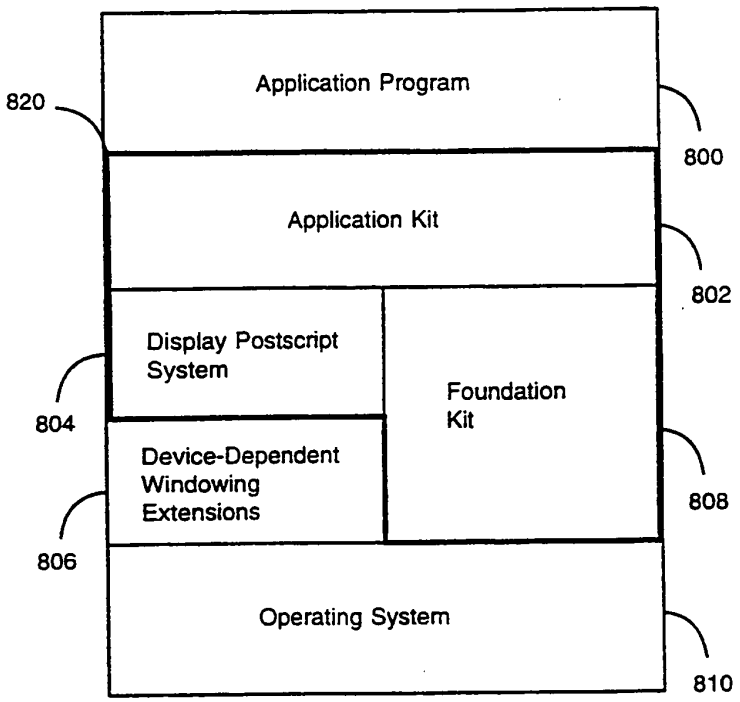
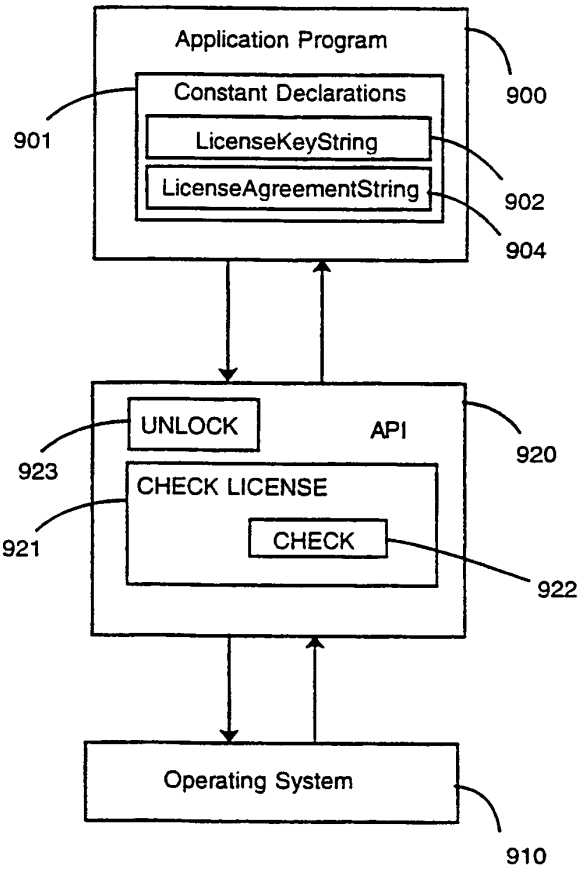

Application Program
820
800
Application Kit
802
Display Postscript System
804
Foundation Kit
Device-Dependent Windowing Extensions
806
808
Operating System
810

FIG. 9


Application Program
900
Constant Declarations
901
LicenseKeyString
902
LicenseAgreementString
904
UNLOCK
API
920
923
CHECK LICENSE
921
CHECK
922
Operating System
910

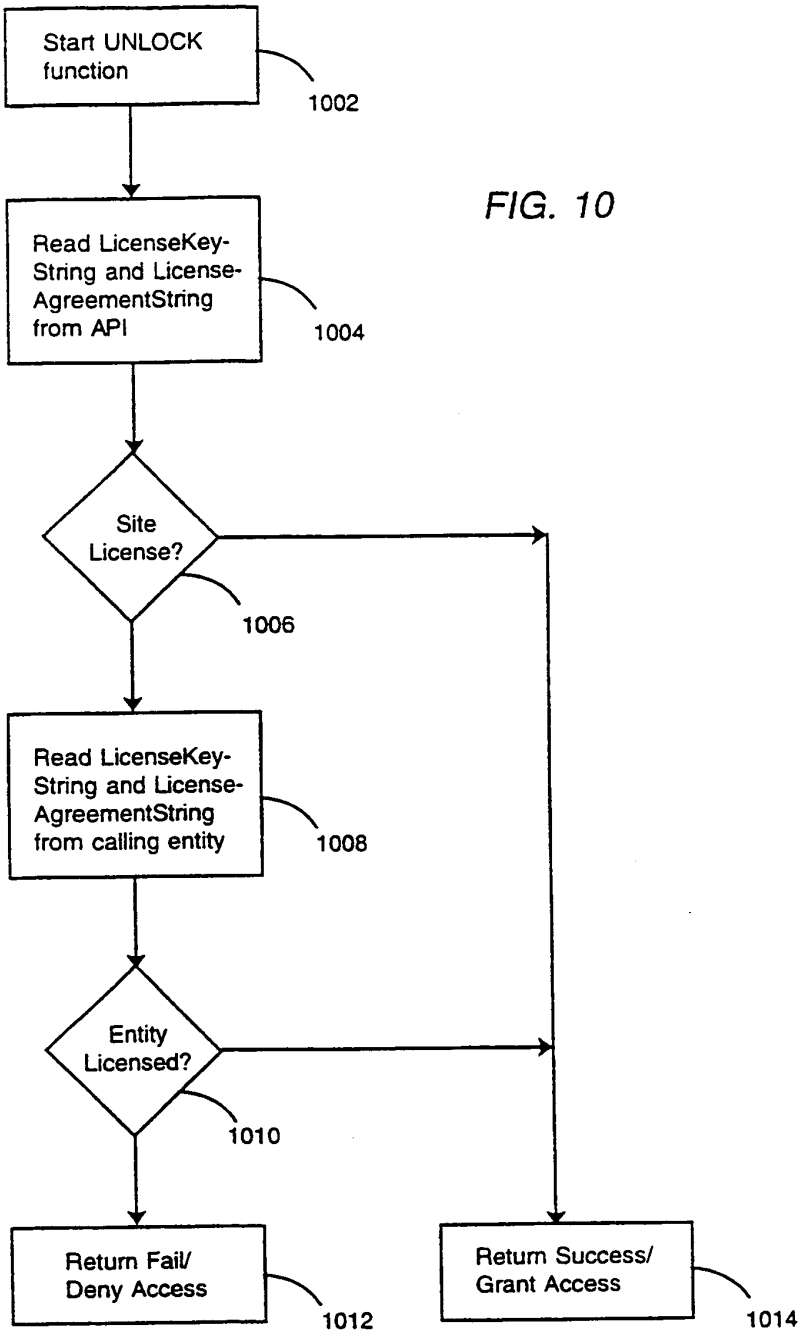

Start UNLOCK function
1002
Read LicenseKey-String and License-AgreementString from API
1004
Site License?
1006
Read LicenseKey-String and License-AgreementString from calling entity
1008
Entity Licensed?
1010
Return Fail/ Deny Access
1012
Return Success/ Grant Access
1014

FIG. 10

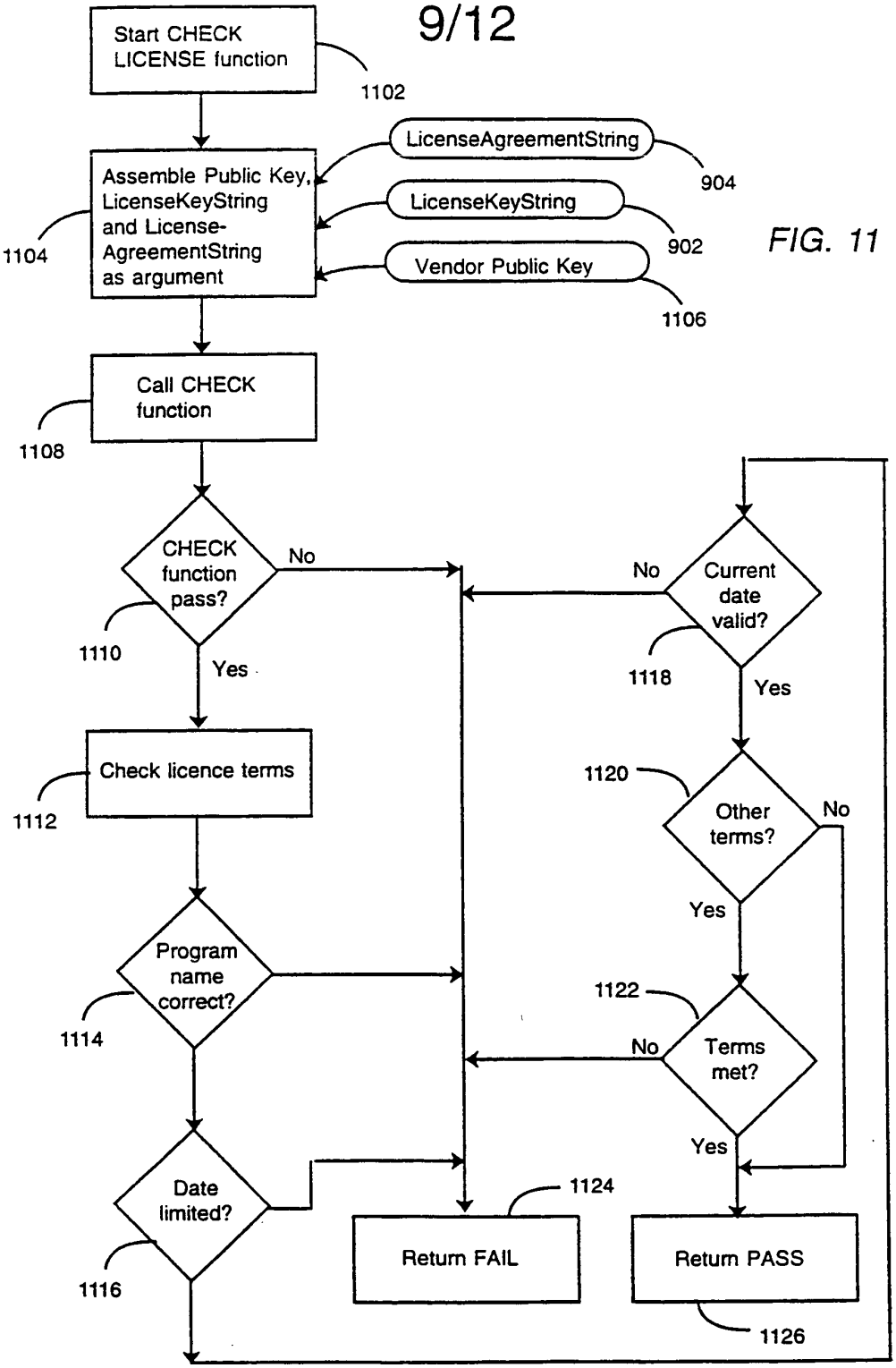

Start CHECK LICENSE function
1102
LicenseAgreementString
904
Assemble Public Key, LicenseKeyString and License-AgreementString as argument
1104
LicenseKeyString
902
Vendor Public Key
1106
Call CHECK function
1108
CHECK function pass?
1110
No
Yes
Check licence terms
1112
Program name correct?
1114
Date limited?
1116
Current date valid?
1118
No
Yes
1120
Other terms?
No
Yes
1122
Terms met?
No
Yes
1124
Return FAIL
Return PASS
1126

FIG. 11

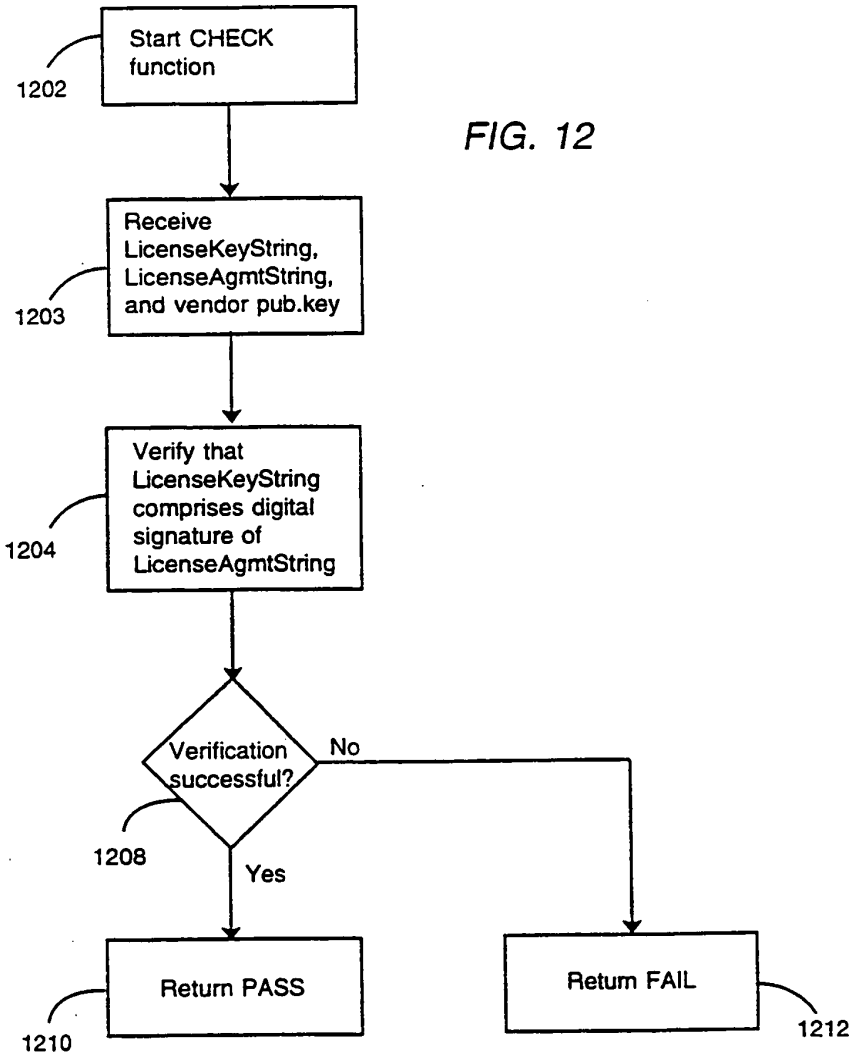
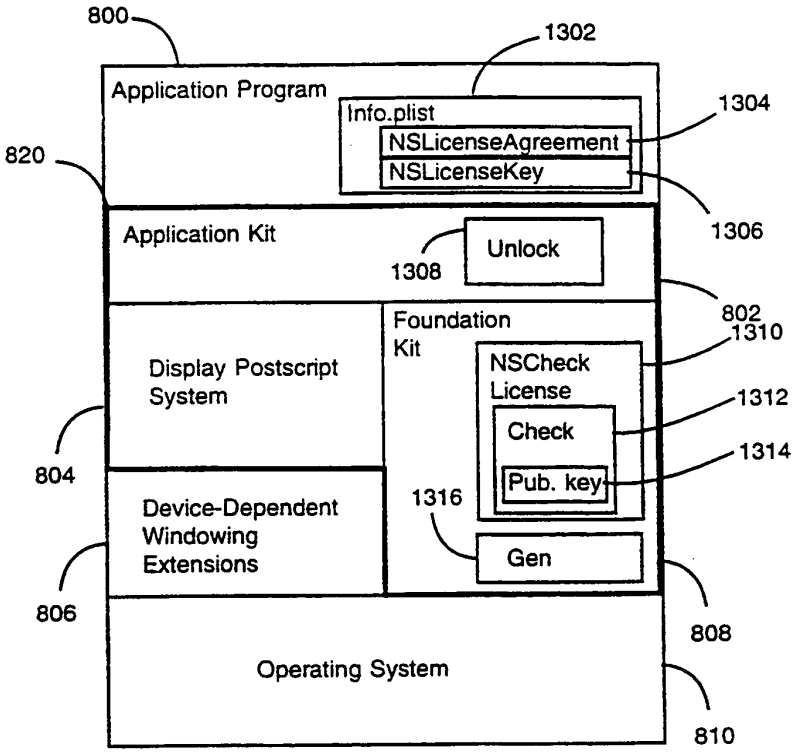

Start CHECK function
1202
Receive LicenseKeyString, LicenseAgmtString, and vendor pub.key
1203
Verify that LicenseKeyString comprises digital signature of LicenseAgmtString
1204
Verification successful?
1208
No
Yes
Return PASS
1210
Return FAIL
1212

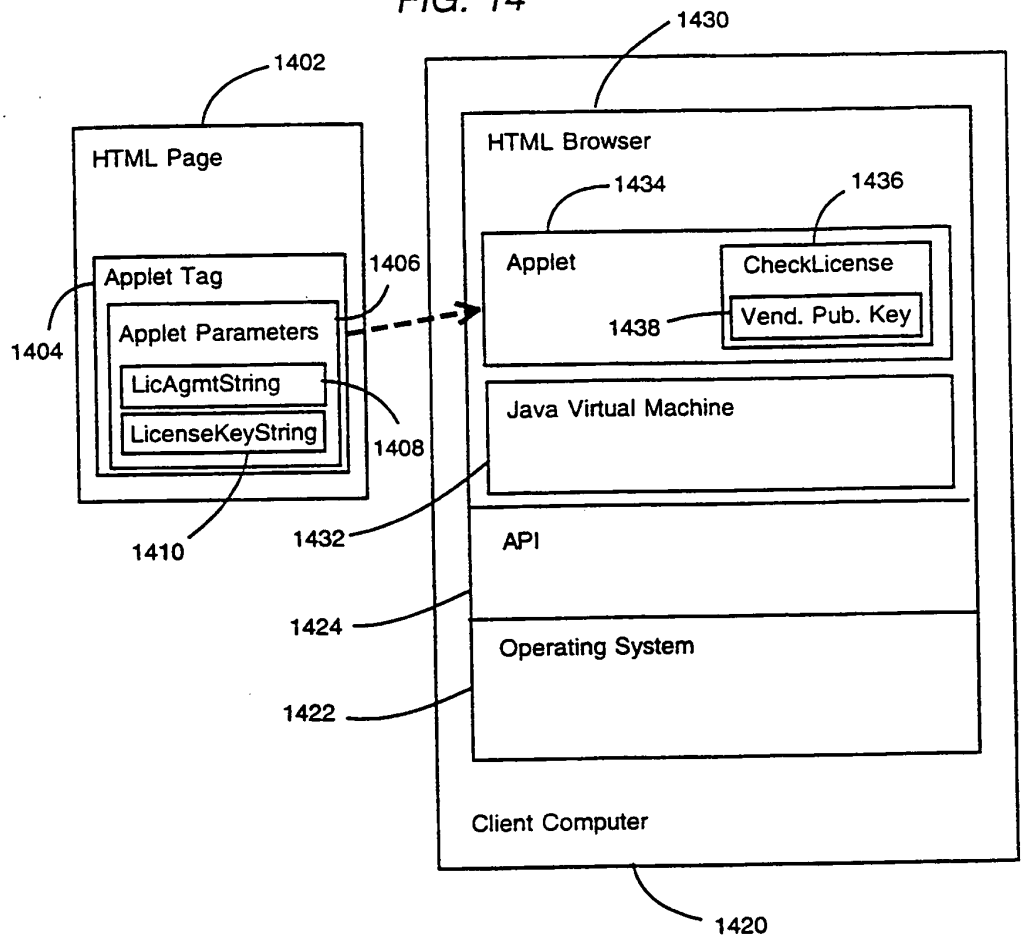
FIG. 12

FIG. 13


800
1302
Application Program
Info.plist
1304
NSLicenseAgreement
NSLicenseKey
820
1306
Application Kit
1308
Unlock
802
Foundation
Kit
1310
NSCheck
License
Display Postscript
System
1312
Check
1314
Pub. key
804
1316
Device-Dependent
Windowing
Extensions
Gen
806
808
Operating System
810

12/12

FIG. 14


1430
1402
HTML Page
HTML Browser
1434
1436
Applet Tag
1406
Applet
CheckLicense
Applet Parameters
1438
Vend. Pub. Key
1404
LicAgmtString
LicenseKeyString
1408
Java Virtual Machine
1410
1432
API
1424
Operating System
1422
Client Computer
1420

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 1/00, 9/46

A3

(11) International Publication Number: WO 99/05600

(43) International Publication Date: 4 February 1999 (04.02.99)

(21) International Application Number: PCT/US98/15340

(22) International Filing Date: 24 July 1998 (24.07.98)

(30) Priority Data:
08/901,776 28 July 1997 (28.07.97) US

(71) Applicant: APPLE COMPUTER, INC. [US/US]; Law Dept., M/S: 38-PAT, 1 Infinite Loop, Cupertino, CA 95014 (US).

(72) Inventors: GARST, Blaine; 3307 Bay Court, Belmont, CA 94002 (US). SERLET, Bertrand; 218 Colorado Avenue, Palo Alto, CA 94301 (US).

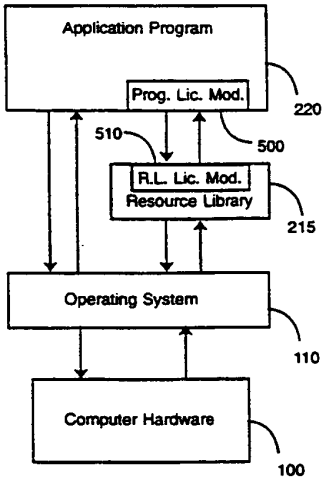
(74) Agents: HECKER, Gary, A. et al.; Hecker & Harriman, Suite 2300, 1925 Century Park East, Los Angeles, CA 90067 (US).

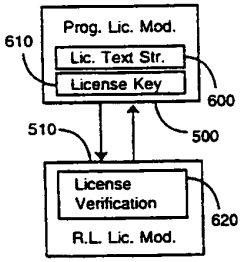
(81) Designated States: CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(88) Date of publication of the international search report:
14 May 1999 (14.05.99)

(54) Title: METHOD AND APPARATUS FOR ENFORCING SOFTWARE LICENSES





(57) Abstract

The present invention comprises a method and apparatus for enforcing software licenses for resource libraries such as an application program interface (API), a toolkit, a framework, a runtime library, a dynamic link library (DLL), an applet (e.g. a Java or ActiveX applet), or any other reusable resource. The present invention allows the resource library to be selectively used only by authorized end user software programs. The present invention can be used to enforce a "per-program" licensing scheme for a resource library whereby the resource library is licensed only for use with particular software programs. In one embodiment, a license text string and a corresponding license key are embedded in a program that has been licensed to use a resource library. The license text string and the license key are supplied, for example, by a resource library vendor to a program developer who wants to use the resource library with an end user program being developed. The license text string includes information about the terms of the license under which the end user program is all wed to use the resource library. The license key is used to authenticate the license text string. The resource library in turn is provided with means for reading the license text string and the license key, and for determining, using the license key, whether the license text string is authentic and whether the license text string has been altered. Resource library functions are made available only to a program having an authentic and unaltered license text string.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	N	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

INTE..ATIONAL SEARCH REPORT

Inte onal Application No
PCT/US 98/15340

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F1/00 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	EP 0 667 572 A (IBM) 16 August 1995 see abstract; figure 4 see page 4, line 53 - page 5, line 27 see claims 1-9	1-83
Y	EP 0 570 123 A (FISCHER ADDISON M) 18 November 1993 see abstract; figure 3D see claims 1-57	1-83
A	WO 97 14087 A (ERICKSON JOHN S) 17 April 1997 see abstract; figures 1,4,10,12 see page 7, paragraph 2 - page 9, paragraph 1	1-83
A	EP 0 778 512 A (SUN MICROSYSTEMS INC) 11 June 1997	

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
12 March 1999	19/03/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Powell, D

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNAT NAL SEARCH REPORT

.nformation on patent family members

Inte: onal Application No
PCT/US 98/15340

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0667572 A	16-08-1995	JP 7230380 A	29-08-1995
		US 5673315 A	30-09-1997
EP 0570123 A	18-11-1993	US 5412717 A	02-05-1995
		AU 3820993 A	18-11-1993
		CA 2095087 A	16-11-1993
		JP 6103058 A	15-04-1994
		US 5311591 A	10-05-1994
WO 9714087 A	17-04-1997	US 5765152 A	09-06-1998
		AU 7662496 A	30-04-1997
EP 0778512 A	11-06-1997	US 5708709 A	13-01-1998
		JP 9288575 A	04-11-1997

Form PCT/ISA/210 (patent family annex) (July 1992)

(19) Europäisches Patentamt
European Patent Office
Office européen des br v ts

(11) EP 0 930 793 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
21.07.1999 Bulletin 1999/29

(51) Int Cl.[6]: **H04Q 7/32**, H04B 1/38, G06F 9/38

(21) Application number: **98310312.8**

(22) Date of filing: **16.12.1998**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **22.12.1997 US 995606**

(71) Applicant: **TEXAS INSTRUMENTS INC.**
Dallas, Texas 75243 (US)

(72) Inventors:
- **McMahon, Michael (NMI)**
  Plano, Texas 75074 (US)
- **Lineberry, Marion C.**
  Dallas, Texas 75218 (US)
- **Woolsey, Matthews A.**
  Plano, Texas 75023 (US)
- **Chauvel, Gerard (NMI)**
  06600 Antibes (FR)

(74) Representative: **Potter, Julian Mark et al**
D. Young & Co.,
21 New Fetter Lane
London EC4A 1DA (GB)

(54) Mobile equipment with a plurality of processors

(57) A wireless data platform (10) comprises a plurality of processors (12, 16). Channels of communication are set up between processors such that they may communicate information as tasks are performed. A dynamic cross compiler (80) executed on one processor compiles code into native processing code for another processor. A dynamic cross linker (82) links the compiled code for other processor. Native code may also be downloaded to the platform through use of a JAVA Bean (90) (or other language type) which encapsulates the native code. The JAVA Bean can be encrypted and digitally signed for security purposes.

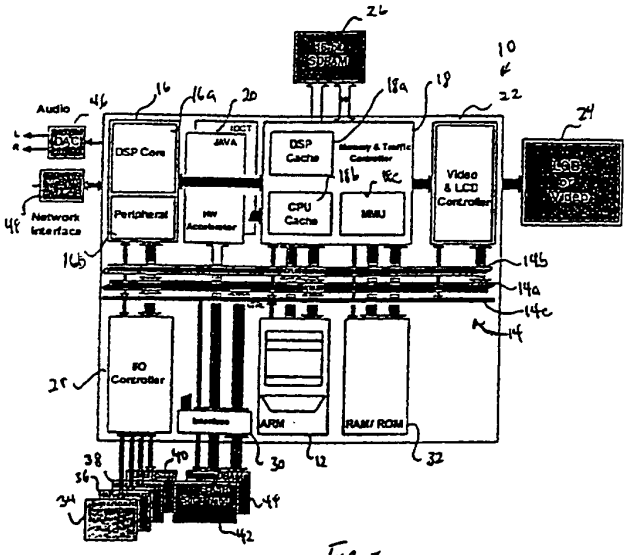


FIG 1

EP 0 930 793 A1

Printed by Jouve, 75001 PARIS (FR)

Description

BACKGROUND OF THE INVENTION

TECHNICAL FIELD

[0001] This invention relates in general to mobile electronic devices and, more particularly, to a hardware and software platform for mobile electronic devices.

DESCRIPTION OF THE RELATED ART

[0002] Handheld portable devices are gaining popularity as the power and, hence, functionality of the devices increases. Personal Digital Assistants (PDAs) are currently in widespread use and Smartphones, which combine some of the capabilities of a cellular phone and a PDA, are expected to have a significant impact on communications in the near future.

[0003] Some devices currently incorporate one or more DSPs (digital signal processor) or other coprocessors for providing certain discrete features, such as voice recognition, and a general purpose processor for other data processing functions. The code for the DSP and the code for the general purpose processor is generally stored in ROMs or other nonvolatile memories, which are not easily modified. Thus, as improvements and new features become available, it is often not possible to upgrade the capabilities of the device. In particular, it is not possible to maximize the use of the DSPs or other coprocessor which may be present in the device.

[0004] Therefore, a need exists for a data processing architecture which can be upgraded and optimizes use of multiple processors and coprocessors.

BRIEF SUMMARY OF THE INVENTION

[0005] The teachings of the present application disclose a mobile electronic device that comprises a coprocessor for executing native code, a host processor system operable to execute native code corresponding to the host processor system and processor independent code. The host processor system is operable to dynamically change the tasks performed by the digital signal coprocessor. Communication circuitry provides for communication between the host processor system and the coprocessor.

[0006] This mobile electronic device significant advantages over the prior art. Because the host processor system can dynamically allocate the tasks being performed by the coprocessor, which may be a digital signal processor, to fully use the coprocessor. The host processor system may direct a routine to one of a plurality of coprocessors, depending upon a variety of factors, such the present capabilities of each processor.

BRIEF DESCRIPTION OF SEVERAL VIEWS OF THE DRAWINGS

[0007] For a more complete understanding of the present invention, and the advantages thereof, reference is now made to the following descriptions taken in conjunction with the accompanying drawings, in which:

Figure 1 illustrates a block diagram of a platform architecture particularly suited for general wireless data processing;

Figure 2 illustrates a functional block diagram of the platform of Figure 1;

Figure 3 illustrates a functional block diagram of dynamic cross compiling and dynamic cross linking functions;

Figure 4 illustrate an embodiment of native code for execution on a processor being encapsulated in a JAVA Bean wrapper for downloading to a device;

Figure 5 illustrates the operation of transferring the encapsulated native code to a processor on a device from a JAVA Bean located on a remote server; and

Figure 6 illustrates a flow diagram describing security features associated with the operation of Figure 5.

DETAILED DESCRIPTION OF THE INVENTION

[0008] Figure 1 illustrates a preferred embodiment of a general wireless data platform architecture, which could be used for example, in the implementation of a Smartphone or PDA. The wireless data platform 10 includes a general purpose (Host) processor 12 coupled to bus structure 14, including data bus 14a, address bus 14b and control bus 14c. One or more DSPs (or other coprocessors) 16, including the core processor 16a and the peripheral interface 16b, are coupled to bus 14 and to memory and traffic controller 18, which includes a DSP cache memory 18a, a CPU cache 18b, and a MMU (memory management unit) 18c. Hardware accelerator circuit 20 (for accelerating a portable language such as JAVA) and a video and LCD controller 22 are also coupled to the memory and traffic controller 18. The output of the video and LCD controller is coupled to an LCD or video display 24.

[0009] Memory & traffic controller 18 is coupled to bus 14 and to the main memory 26, shown as an SDRAM (synchronous dynamic random access memory). Bus 14 is also connected to I/O controller 28, interface 30, and RAM/ROM 32. A plurality of devices could be coupled to the wireless data platform 10, such as smartcard 34, keyboard 36, mouse 38, or one or more serial ports 40, such as a USB (universal serial bus) port or an RS232 serial port. Interface 30 can couple to a flash memory card 42 and/or a DRAM card 44. The peripheral interface 16b can couple the DSP 16 to a DAC (digital to analog converter) 46, a network interface 48 or to other devices.

[0010] The wireless data platform 10 of Figure 1 utilizes both a general purpose processor 12 and a DSP 16. Unlike current devices in which the DSP 16 is dedicated to specific fixed functions, the DSP 16 of Figure 1 can be used for any number of functions. This allows the user to derive the full benefit of the DSP 16.

[0011] One main area in which the DSP 16 can be used is in connection with the man-machine interface (MMI). Importantly, functions like speech recognition, image and video compression and decompression, data encryption, text-to-speech conversion, and so on, can be performed much more efficiently using the DSP 16. The proposed architecture allows new functions and enhancements to be easily added to wireless data platform 10.

[0012] It should be noted that the wireless data platform 10 is a general block diagram and many modifications could be made. For example, Figure 1 illustrates separate DSP and processor caches 18a and 18b. As would be known to one skilled in the art, a unified cache could also be used. Further, the hardware acceleration circuit 20 is an optional item. Such devices speed the execution of languages such as JAVA; however, the circuit is not necessary for operation of the device. Further, although the illustrated embodiment shows a single DSP, multiple DSPs (or other coprocessors) could be coupled to the buses.

[0013] Figure 2 illustrates a functional software architecture for the wireless data platform 10. This block diagram presumes the use of JAVA; it should be noted that languages other than JAVA could be used as well. Functionally, the software is divided into two groups, Host processor software and DSP software. The Host software includes one or more applets 41 . The DSP API class 43 is a JAVA API package for JAVA applications or applets to access the functionality of the DSP API 50 and Host DSP Interface Layer 52. A JAVA virtual machine (VM) 45 interprets the applets. The JAVA native interface 47 is the method which the JAVA VM executes host processor or platform specific code. Native tasks 49 are non-JAVA programs which can be executed by the Host processor 12 without using the JAVA native interface. The DSP API 50, described in greater detail hereinbelow, is an API (application program interface) used the Host 12 to call to make use of the capabilities of the DSP 16. The Host-DSP Interface Layer 52 provides an API for the Host 12 and DSP 16 to communicate with each other, with other tasks, or other hardware using channels via the Host-DSP Communication Protocol. The DSP device driver 54 is the Host based device driver for the Host RTOS 56 (real time operating system) to communicate with the DSP 16. The Host RTOS 56 is an operating system, such as NUCLEUS PLUS by Accelerated Technology Incorporated.

[0014] Alternatively a non-real time operating system, such as WINDOWS CE by Microsoft Corporation, could b used. The DSP Library 58 contains programs stored for execution on the DSP 16.

[0015] On the DSP side, one or more tasks 60 can be stored in memory for execution by the DSP16. As described below, the tasks can be moved in and out of the memory as desired, such that the functionality of the DSP is dynamic, rather than static. The Host-DSP Interface layer 62 on the DSP side performs the same function as the Host-DSP Interface layer 52 on the Host side, namely it allows the Host 12 and DSP 16 to communicate. The DSP RTOS 64 is the operating system for the DSP processor. The Host Device driver 66 is a DSP based device driver for the DSP RTOS 64 to communicate with the Host 12. The Host-DSP Interface 70 couples the DSP 16 and Host 12.

[0016] In operation, the software architecture shown in Figure 2 uses the DSP 16 as a variable function device, rather than a fixed function device as in the prior art.

[0017] Accordingly, the DSP functions can be downloaded to the mobile device incorporating the architectur of Figur 2 to allow the DSP 16 to perform various signal processing functions for the Host 12.

[0018] Th DSP-API provides a device indep ndent interfac from the Host 12 to th DSP 16. Th functions provide the Host 12 with th ability to load and schedule tasks on th DSP 16 and to control and communicat with those tasks. Th API functions include calls to: determine th DSP's available resources, creat and control Host 12 and DSP tasks, create and control data channels between Host 12 and DSP tasks, and communicat with tasks. These functions are

described below. Each function returns a BOOLean result, which will be SUCCESS for a successful operation, or FAILURE. If the result is FAILURE, the errcode should be checked to determine which error occurred.

DSP_G t_MIPS

*BOOL DSP_Get_MIPS(T_DeviceID DevID, U32 *mips, U16 *errcode);*

[0019] This function returns the current MIPS available on the DSP. This consists of the MIPS capability of the DSP 16 minus a base MIPS value (the MIPS value with no additional dynamic tasks, i.e. the kernel plus API code plus drivers), minus the sum of the MIPS ratings for all loaded dynamic tasks. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING

DSP_Get_Memory_Available

*BOOL DSP_Get_Memory_Available(T_DeviceID DevID, T_Size *progmem, T_Size *datamem, U16 *errcode);*

[0020] This function will query the DSP 16 specified by *DevID* for the amounts of available memory for both program memory and data memory. The resultant values are returned in the *progmem* and *datamem* parameters. The sizes are specified in T_DSP_Words. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING

DSP_Alloc_Mem

*BOOL DSP_Alloc_Mem(T_DeviceID DevID, U16 mempage, T_Size size, T_DSP_Word **memptr, U16 *errcode);*

[0021] This function will allocate a block of memory on a DSP 16. The *DevID* specifies which device on which to allocate the memory. The *mempage* is 0 for program space, and 1 for data space. The *size* parameter specifies the memory block size in T_DSP_Words. The returned *memptr* will be a pointer to the memory block on the DSP 16, or NULL on failure. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE
DSP_NOT_ENOUGH_MEMORY

DSP_Free_Mem

*BOOL DSP_Free_Mem(T_DeviceID DevID, U16 mempage, T_DSP_Word *memptr, U16 *errcode);*

[0022] This function will free a block of memory on a DSP that was allocated with the DSP_Alloc_Mem function. The *DevID* specifies on which device the memory resides. The *mempage* is 0 for program space, and 1 for data space. The *memptr* parameter is the pointer to the memory block. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE
DSP_MEMBLOCK_NOT_FOUND

DSP_Get_Code_Info

*BOOL DSP_Get_Code_Info(char *Name, T_CodeHdr *codehdr, U16 *errcode);*

[0023] This function will access the DSP Library table and return the code header for the DSP function code specified by the *Name* parameter. On return, the location pointed to by the *codehdr* parameter will contain the code header information. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_NAMED_FUNC_NOT_FOUND

DSP_Link_Code

*BOOL DSP_Link_Code(T_DeviceID DevID, T_CodeHdr *codehdr, T_TaskCreate *tcs, U16 *errcode);*

[0024] This function will link DSP function code so that it will run at a specified address on the DSP specified by *DevID*. The *codehdr* param ter points to th code header for th function. Th dynamic cross link r will link th code bas d on information in the cod h ader, and in th cod (COFF fil). The dynamic cross link r will allocat th memory as n eded, and link and load th code to the DSP 16. Th *tcs* param t r is a point r to th task creation structure needed in the DSP_Cr ate_Task function. DSP_Link_Code will fill in th code ntry points, priority, and quantum fi lds of the structur in preparation for creating a task. Th *errcode* paramet r will contain the following possible results:

DSP_SUCCESS

DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_NOT_ENOUGH_PROG_MEMORY
DSP_NOT_ENOUGH_DATA_MEMORY
DSP_COULD_NOT_LOAD_CODE

DSP_Put_BLOB

*BOOL DSP_Put_BLOB(T_DeviceID DevID, T_HostPtr srcaddr, T_DSP_Ptr destaddr, U16 mempage, T_Size size, U16 *errcode);*

[0025] This function will copy a specified Binary Large Object (BLOB) to the DSP 16. The *DevID* specifies on which DSP 16 to copy the object. The *srcaddr* parameter is a pointer to the object in Host memory. The *destaddr* is a pointer to the location to which to copy the object on the DSP 16. The *mempage* is 0 for program space, and 1 for data space. The size parameter specifies the size of the object in T_DSP_Words. The *errcode* parameter will contain the following possible results :

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

DSP_Create_Task

*BOOL DSP_Create_Task(T_DeviceID DevID, T_TaskCreate *tcs, T_TaskID *TaskID, U16 *errcode);*

[0026] DSP_Create_Task requests the DSP 16 to create a task given the task parameters and the code locations in the DSP's program space. The Task Creation Structure is show in Table 1:

Table 1.

Task Creation Structure.		
Data Type	**Field Name**	**Description**
T_DSP_Name	Name	User defined name for the task.
U32	MIPS	MIPS used by the task.
T_ChanID	ChanIn	The channel ID used for task input.
T_ChanID	ChanOut	The channel ID used for task output
T_StrmID	StrmIn	The stream ID used for task input
T_StrmID	StrmOut	The stream ID used for task output.
U16	Priority	The task's priority.
U32	Quantum	The task's timeslice in system ticks.
T_Size	StackReq	The amount of stack required.
T_DSP_Ptr	MsgHandler	Pointer to code to handle messages to the task.
T_HOST_Ptr	CallBack	Pointer to Host code to handle messages from the task.
T_DSP_Ptr	Create	Pointer to code to execute when task is created.
T_DSP_Ptr	Start	Pointer to code to execute when task is started.
T_DSP_Ptr	Suspend	Pointer to code to execute when task is suspended.
T_DSP_Ptr	Resume	Pointer to code to execute when task is resumed.
T_DSP_Ptr	Stop	Pointer to code to execute when task is stopped.

[0027] Once the task is created, the Create entry point will be called, giving the task the opportunity to do any necessary preliminary initialization. The Create, Suspend, Resume, and Stop entry points can be NULL. The resultant *TaskID* contains both the device ID (DevID), and the DSP's task ID. If the *TaskID* is NULL, the create failed. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_PRIORITY
DSP_CHANNEL_NOT_FOUND
DSP_ALLOCATION_ERROR

DSP_Start_Ta k

*BOOL DSP_Start_Task(T_TaskID TaskID, U16 *errcode);*

[0028] This function will start a DSP task sp cified by *TaskID.* Execution will b gin at th task's Start ntry point. The

errcode parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

DSP_Suspend_Task

*BOOL DSP_Suspend_Task(T_TaskID TaskID, U16 *errcode);*

[0029] This function will suspend a DSP task specified by *TaskID.* Prior to being suspended, the task's Suspend entry point will be called to give the task a chance to perform any necessary housekeeping. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

DSP_Resume_Task

*BOOL DSP_Resume_Task(T_TaskID TaskID, U16 *errcode);*

[0030] This function will resume a DSP task that was suspended by DSP_Suspend_Task. Prior to being resumed, the task's Resume entry point will be called to give the task a chance to perform any necessary housekeeping. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND
DSP_TASK_NOT_SUSPENDED

DSP_Delete_Task

*BOOL DSP_Delete_Task(T_TaskID TaskID, U16 *errcode);*

[0031] This function will delete a DSP task specified by *TaskID.* Prior to the deletion, the task's Stop entry point will be called to give the task a chance to perform any necessary cleanup. This should include freeing any memory that was allocated by the task, and returning any resources the task acquired. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

DSP_Change_Task_Priority

*BOOL DSP_Change_Task_Priority(T_TaskID TaskID, U16 newpriority, U16 *oldpriority, U16 *errcode);*

[0032] This function will change the priority of a DSP task specified by *TaskID.* The priority will be changed to *newpriority.* The possible values of *newpriority* are RTOS dependent. Upon return, the *oldpriority* parameter will be s t to the previous priority of the task. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND
DSP_INVALID_PRIORITY

DSP_Get_Task_Status

*BOOL DSP_Get_Task_Status(T_TaskID TaskID, U16 *status, U16 *priority, T_ChanID *Input, T_ChanID *Output, U16 *errcode);*

[0033] This function returns the status for a DSP task specified by *TaskID.* The *status* will be one of the following values:

DSP_TASK_RUNNING
DSP_TASK_SUSPENDED
DSP_TASK_WAITFOR_SEM
DSP_TASK_WAITFOR_QUEUE
DSP_TASK_WAITFOR_MSG

[0034] The *priority* parameter will contain th task's priority, and the *Input* and *Output* paramet rs will contain the task's input and output chann l IDs, respectively. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND

DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

DSP_Get_ID_From_Name

*BOOL DSP_Get_ID_From_Name(T_DeviceID DevID, T_DSP_Name Name, T_DSP_ID *ID, U16 *errcode);*

[0035] This function returns the ID for a named object on the DSP 16. The named object may be a channel, a task, a memory block, or any other supported named DSP object. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_NAME_NOT_FOUND

DSP_Dbg_Read_Mem

*BOOL DSP_Dbg_Read_Mem(DEVICE_ID DevID, U8 mempage, DSP_PTR addr, U32 count, DSP_WORD *buf, U16 *errcode);*

[0036] This function requests a block of memory. The *mempage* specifies program memory (0) or data memory (1). The *addr* parameter specifies the memory starting address, and the *count* indicates how many T_DSP_Words to read. The bufparameter is a pointer to a caller provided buffer to which the memory should be copied. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

DSP_Dbg_Write_Mem

*BOOL DSP_Dbg_Write_Mem(T_DeviceID DevID, U16 mempage, T DSP_Ptr addr, T_Count count, T_DSP_Word *buf, U16 *errcode);*

[0037] This function writes a block of memory. The *mempage* specifies program memory (0) or data memory (1). The *addr* parameter specifies the memory starting address, and the *count* indicates how many T_DSP_Words to write. The *buf* parameter is a pointer the buffer containing the memory to write. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

DSP_Dbg_Read_Reg

*BOOL DSP_Dbg_Read_Reg(T_DeviceID DevID, U16 RegID, T_DSP_Word *regvalue, U16 *errcode);*

[0038] This function reads a DSP register and returns the value in *regvalue.* The *RegID* parameter specifies which register to return. If the *RegID* is -1, then all of the register values are returned. The *regvalue* parameter, which is a pointer to a caller provided buffer, should point to sufficient storage to hold all of the values. The register IDs are DSP specific and will depend on a particular implementation. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_REGISTER

DSP_Dbg_Write_Reg

*BOOL DSP_Dbg_Write_Reg(T_DeviceID DevID, U16 RegID, T_DSP_Word regvalue, U16 *errcode);*

[0039] This function writes a DSP register. The *RegID* parameter specifies which register to modify. *regvalue* contains the new value to write. The register IDs are DSP specific and will depend on a particular implementation. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_REGISTER

DSP_Dbg_Set_Break

*BOOL DSP_Dbg_Set_Break(T_DeviceID DevID, DSP_Ptr addr, U16 *errcode);* This function sets a br ak point at the giv n cod addr ss *(addr).* Th *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND

DSP_DEVID_NOT_RESPONDING

DSP_Dbg_Clr_Break

*BOOL DSP_Dbg_Clr_Break(T_DeviceID DevID, T_DSP_Ptr addr, U16 *errcode);*

[0040] This function clears a break point that was previously set by DSP_Dbg_Set_Break at the given code address *(addr)*. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_BP_DID_NOT_EXIST

[0041] The DSP Device Driver 54 handles communications from the Host 12 to the DSP 16. The driver functions will take the communication requests as specified in the Host-DSP Communications Protocol and handle the transmission of the information via the available hardware interface. The device driver is RTOS dependent and communications hardware dependent.

[0042] The DSP Library 58 contains the blocks of code that can be downloaded to the DSP 16 for execution. Each block of code will be previously unlinked, or relocatably linked as a library, so that the dynamic cross linker can resolve all address references. Each code block will also include information about the block's requirements for DSP MIPS (millions of instructions per second), priority, time slice quantum, and memory. The format for the code block header is shown in Table 2. The program memory and data memory sizes are approximations to give the Host 12 a quick check on whether the DSP can support the task's memory requirements. If there appears to be sufficient space, the dynamic cross linker can then attempt to link and load the code. It should be noted that the dynamic cross linker could still fail, due to page alignment and contiguity requirements. In the preferred embodiment, the code is in a version 2 COFF file format.

Table 2.

Code Block Header.		
Data Type	Field Name	Description
U16	Processor	The target processor type.
T_DSP_Name	Name	Task's name.
U32	MIPS	Worst case MIPS required by the task.
T_Size	ProgSize	Total program memory size needed.
T_Size	DataSize	Total data memory size needed.
T_Size	InFrameSize	Size of a frame in the task's input channel.
T_Size	OutFrameSize	Size of a frame in the task's output channel.
T_Size	InStrmSize	Size of the task's input stream FIFO.
T_Size	OutStrmSize	Size of the task's output stream FIFO.
U16	Priority	Task's priority.
U32	Quantum	Task's time slice quantum (number of system ticks).
T_Size	StackReq	Stack required.
T_Size	CoffSize	Total size of the COFF file.
T_DSP_Ptr	MsgHandler	Offset to a message handler entry point for the task.
T_DSP_Ptr	Create	Offset to a create entry point that is called when the task is created.
T_DSP_Ptr	Start	Offset to the start of the task's code.
T_DSP_Ptr	Suspend	Offset to a suspend entry point that is called prior to the task being suspended.
T_DSP_Ptr	Resume	Offset to a resume entry point that is called prior to the task being resumed.
T_DSP_Ptr	Stop	Offset to a stop entry point that is called prior to the task being deleted.
T_Host_Ptr	CoffPtr	Pointer to the location of the COFF data in the DSP Library.

[0043] A procedure for converting portable (processor independent) code, such as JAVA code, into linked target code is shown in Figure 3. The procedure uses two functions, a dynamic cross compiler 80 and a dynamic cross linker 82. Each function is implemented on the host processor 12. The dynamic cross linker is part of the DSP-API in the preferred embodiment. The cross compiler may also be part of the DSP-API.

[0044] The dynamic cross compiler 80 converts portable code into unlinked, executable target processor code. The dynamic cross linker 82 converts the unlinked, executable target processor code into linked, executable target processor code. To do so, it must resolve addresses within a block of code, prior to loading on the DSP 16. The dynamic

cross linker 82 links the code segments and data segments of the function, allocates the memory on the DSP 16, and loads the code and constant data to the DSP 16. The functions are referred to as "cross" compiling and "cross" linking, because the functions (compiling and linking) occur on a different processor (i.e., the host processor 12) from the target processor which executes the code (i.e., the DSP 16).

[0045] The dynamic cross compiler 80 accepts previously unlinked code loaded on demand by a user or a user agent (such as a browser). The code is processed to either (1) identify "tagged" sections of the code or (2) analyze untagged code segments for suitability of execution on the DSP 16. A tagged section of source code could delineate source targetable to a DSP by predetermined markers such as "<start DSP code>" and <end DSP code>" embedded in the source code. If a tagged section is identified either directly or through analysis, a decision is made to either cross compile or not based on the current processing state of the DSP 16. If a decision is made to compile, the section of code processed by compiling software that outputs unlinked, executable target processor code, using well known compiling methods. A decision not to compile could be made if for example, the DSP has insufficient available processing capacity (generally stated as available MIPS - million of instructions per second) or insufficient available memory, due to other tasks being executed by the DSP 16. The compiled code can be passed to the dynamic cross linker 82 for immediate use in the DSP 16, or could be saved in the DSP library 58.

[0046] The dynamic cross linker 82 accepts previously unlinked code, which is either (1) statically stored in connection with the host processor 12 or (2) dynamically downloaded to the host processor 12 over a network connection (including global networks such as the Internet) or (3) dynamically generated by the dynamic cross compiler 80. The dynamic cross linker 82 links the input code for a memory starting address of the DSP 16 determined at runtime. The memory starting address can be determined from a memory map or memory table stored on and managed by either the host processor 12 or DSP 16. The dynamic cross linker 82 convert referenced memory locations in the code to actual memory locations in the DSP 16. These memory locations could include, for example, branch addresses in the code or references to locations of data in the code.

[0047] In the preferred embodiment, the portable code is in a COFF (common object file format) which contains all information about the code, including whether it is linked or unlinked. If it is unlinked, symbol tables define the address which must be changed for linking the code.

[0048] The conversion process described above has several significant advantages over the prior art. First, the dynamic cross compiler 80 allows run-time decisions to be made about where to execute the downloaded portable code. For example, in a system with multiple target processors (such as two DSPs 16), the dynamic cross compiler 80 could compile the portable code to any one of the target processors based on available resources or capabilities. The dynamic cross linker 82 provides for linking code to run on a target processor which does not support relocatable code. Since the code is linked at run-time, memory locations in the DSP 16 (or other target processor) do not need to be reserved, allowing optimum efficiency of use of all computing resources in the device. Because the compiling is accomplished with knowledge of the architecture of the platform 10, the compiling can take advantage of processor and platform specific features, such as intelligent cache architectures in one or both processors 12 and 16.

[0049] Thus, the DSP 16 can have various functions which are changed dynamically to fully use its processing capabilities. For example, the user may wish to 12 load a user interface including voice recognition. At that time, the host processor 12 could download software and dynamically cross compile and cross link the voice recognition software for execution in the DSP 16. Alternatively, previously compiled software in the DSP library 58 could be dynamically cross linked, based on the current status of the DSP 16, for execution.

[0050] The Host Device Driver handles communications from the DSP 16 to the Host Processor 12. The driver functions takes the communication requests as specified in the Host-DSP Communications Protocol and handles transmission of the information via the available hardware interface. The device driver is RTOS dependent and communications hardware dependent.

[0051] The Host-DSP Communications Protocol governs the communications of commands and data between the Host 12 and the DSP 16. The communications consist of several paths: messages, data channels, and streams. Messages are used to send initialization parameters and commands to the tasks. Data channels carry large amounts of data between tasks and between the DSP 16 and Host 12, in the form of data frames. Streams are used to pass streamed data between tasks and between the DSP 16 and Host 12.

[0052] Each task has an entry point to a message handler, which handles messages. The messages are user defined and will include initialization parameters for the task's function, and commands for controlling the task. The tasks send messages to the Host 12 via the callback specified when the task is created. The prototype for the task's message handler and the prototype for the Host's callback are shown here:

```
void TaskMsgHandler(T_ReplyRef replyref, T_MsgID MsgID, T_Count count, T_DSP_Word *buf);
void HostCallBack(T_ReplyRef replyref T_MsgID MsgID, T_Count count, T_DSP_Word *buf);
```

[0053] The *replyref* param ter r fers to an implementation dependent referenc value, which is used to route th

reply back to the sender. For every Send_Message call, the recipient must call Reply_Message using the *replyref* parameter. The actual messages may appear as follows:

Sent message	MsgPktFlag	taskid	replyref	msgid	count	buf[.....]
Reply message	MsgPktFlag	-1	replyref	msgid	count	buf[.....]

The multiword data is sent least-significant word first.

[0054] A *TaskID* of 0 in the Send_Message function indicates a system level message. The system level messages are used to implement the DSP-API functions

[0055] Following are the Message functions:

Send_Message

*BOOL Send_Message(T_TaskID TaskID, T_MsgID MsgID, T_Count count, T_DSP_Word *msgbuf, T_DSP_Word *replybuf, T_Size replybufsize, T_Count replycount, U16 *errcode);*

[0056] This function will send a user defined message to a task specified by *TaskID*. The *MsgID* defines the message, and the *msgbuf* contains the actual message data. The message size is *count* T_DSP_Words. The reply *to* the message will be contained in the *replybuf* parameter, which points to a buffer of size *replybufsize*, provided by the caller. It should be of sufficient size to handle the reply for the particular message. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

Reply_Message

*BOOL Reply_Message(T_ReplyRef replyref, T_Count count, T_DSP_Word *buf, U16 *errcode);*

[0057] This function is used to reply to messages. The *replyref* parameter is a reference used to route the reply back to the sender of the original message, and is implementation specific. The reply is contained in the *buf* parameter and its size is *count* T_DSP_Words. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_BAD_REPLY_REF

[0058] The concept of channels is used to transmit frame-based data from one processor to another, or betw en tasks on the same processor. When created, a channel allocates a specified number and size of frames to contain the data. Initially, the channel will contain a list of empty frames. Tasks that produce data will request empty frames in which to put the data, then once filled, the frame is returned to the channel. Tasks that consume data will request full frames from the channel, and once emptied, the frame is returned to the channel. This requesting and returning of frame buffers allows data to move about with a minimum of copying.

[0059] Each task has a specified Input and Output channel. Once a channel is created, it should be designated as the input to one task, and the output to another task. A channel's ID includes a device ID, so channels can pass data between processors. Channel data flow across the Host-DSP interface may appear as follows:

ChanPktFlag	Channel ID	Count	Data[...]

The following are the channel functions:

Create_Channel

*BOOL Create_Channel(T_DeviceID DevID, T_Size framesize, T_Count numframes, T_ChanID *ChannelID, U16 *errcode);*

[0060] This function creates a data frame-based communication channel. This creates a channel control structure, which maintains control of a set of frame buffers, whose count and size are specified in the *numframes* and *framesize* parameters, respectively. When created, the channel allocates the data frames, and adds them to its list of empty frames. *ChannelID* will return the ID of the new channel. If the *DevID* is not that of the calling processor, a channel control structure is created on both the calling processor and th *DevID* proc ssor, to control data flowing across the communications int rfac . Th *errcode* param ter will contain th following possibl results:

CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING

CHAN_ALLOCATION_ERROR

Delete_Channel

*BOOL Delete_Channel(T_ChanID ChannelID, U16 *errcode);*

[0061] This function deletes an existing channel specified by *ChannelID*. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT RESPONDING
CHAN_CHANNEL_NOT_FOUND

Request_Empty_Frame

*BOOL Request_Empty_Frame(T_LocalChanID Chn, T_DSP_Word **bufptr, BOOL WaitFlag, U16 *errcode);*

[0062] This function requests an empty frame from the specified local channel ID. If *Chn* is NULL, then the task's output channel is used. Upon return, the *bufptr* parameter will contain the pointer to the frame buffer. If the *WaitFlag* is TRUE, and there is no frame buffer available, the caller will be suspended until a buffer becomes available. If the *WaitFlag* is FALSE, the function will return regardless. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_UNAVAILABLE

Return_Full_Frame

*BOOL Return_Full_Frame(T_LocalChanID Chn, T_DSP_Word *bufptr, U16 *errcode);*

[0063] Once a task has filled a frame buffer, it returns is to the channel using this function. The buffer pointed to by *bufptr* is returned to the channel ID specified. If *Chn* is NULL, then the task's output channel is used. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_CTRL_ERROR

Request_Full_Frame

*BOOL Request_Full_Frame(T_LocalChanID Chn, T_DSP_Word **bufptr, BOOL WaitFlag, U16 *errcode);*

[0064] This function requests a full frame of data from the specified local channel ID. If *Chn* is NULL, then the task's input channel is used. Upon return, the *bufptr* parameter will contain the pointer to the frame buffer. If the *WaitFlag* is TRUE, and there are no full frame buffers available, the caller will be suspended until a buffer becomes availabl . If the *WaitFlag* is FALSE, the function will return regardless. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_UNAVAILABLE

Return_Empty_Frame

*BOOL Return_Empty_Frame(T_LocalChanID Chn, T_DSP_Word *bufptr, U16 *errcode);*

[0065] Once a task has used the data from a frame buffer, it should return the buffer to the channel using this function. The buffer pointed to by *bufptr* is returned to the channel ID specified. If *Chn* is NULL, then the task's input chann l is used. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_CTRL_ERROR

Set_Task_Input_Channel

*BOOL Set_Task_Input_Channel(T_Task *TaskID, T_ChanID ChanID, U16 *errcode);*

[0066] This function sets a task's input channel to the specified channel ID. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_TASK_NOT_FOUND
CHAN_CHANNEL_NOT_FOUND

Set_Task_Output_Channel

*BOOL Set_Task_Output_Channel(T_Task *TaskID, T_ChanID ChanID, U16 *errcode);*

[0067] This function s ts a task's output channel to the specified channel ID. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS

CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_TASK_NOT_FOUND
CHAN_CHANNEL_NOT_FOUND

[0068] Streams are used for data, which can not be broken into frames, but which continuously flow into and out of a task. A stream will consist of a circular buffer (FIFO) with associated head and tail pointers to track the data as it flows in and out. Each task can have a designated input and output stream. Stream data flow across the Host-DSP interface may appear as follows:

StrmPktFlag	Stream ID	Count	Data[...]

Following are the stream functions:

Create_Stream

*BOOL Create_Stream(T_DeviceID DevID, T_Size FIFOsize, T_StrmID *StreamID, U16 *errcode);*

[0069] This function creates a FIFO-based communication stream. This creates a stream control structure, which maintains control of a FIFO of size *FIFOsize*. When created, the stream allocates an empty FIFO, and initializes head and tail pointers to handle data flow into and out of the stream. *StreamID* will return the ID of the new stream. If the *DevID* is not that of the calling processor, a stream control structure is created on both the calling processor and the *DevID* processor, to control data flowing across the communications interface. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_ALLOCATION_ERROR

Delete_Channel

*BOOL Delete_Stream(T_StrmID StreamID, U16 *errcode);*

[0070] This function deletes an existing stream specified by *StreamID*. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND

Get_Stream_Count

*BOOL Get_Stream_Count(T_LocalStrmID StrmID, T_Count *count, U16 *errcode);*

[0071] This function requests the count of T_DSP_Words currently in the stream FIFO specified by *StrmID*. The *count* parameter will contain the number upon return. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_STREAM_NOT_FOUND

Write_Stream

*BOOL Write_Stream(T_LocalStrmID Strm, T_DSP_Word *bufptr, T_Count count, T_Count *countwritten, U16 *errcode);*

[0072] This function will write *count* number of T_DSP_Words to the stream specified by the *Strm*. If *Strm* is NULL, the task's output stream is used. The data is pointed to by the *bufptr* parameter. Upon return, *countwritten* will contain the number of T_DSP_Words actually written. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND
STRM_STREAM_OVERFLOW

Read_Stream

*BOOL Read_Stream(T_LocalStrmID Strm, T_DSP_Word *bufptr, T_Count maxcount, BOOL WaitFlag, T_Count *countread, U16 *errcode);*

[0073] This function reads data from the stream specified by *Strm*. If *Strm* is NULL, the task's input stream is used. The data will be stored in the buffer pointed to by *bufptr*. Up to *maxcount* T_DSP_Words will be read from the stream. The *countread* parameter will contain the actual count of the data read. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS

STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND

Set_Task_Input_Str am

*BOOL Set_Task_Input_Stream(T_Task *TaskID, T_StrmID StrmID, U16 *errcode);*

[0074] This function sets a task's input stream to the specified stream ID. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_TASK_NOT_FOUND
STRM_STREAM_NOT_FOUND

Set_Task_Output_Stream

*BOOL Set_Task_Output_Stream(T_Task *TaskID, T_StrmID StrmID, U16 *errcode);*

[0075] This function sets a task's output stream to the specified stream ID. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_TASK_NOT_FOUND
STRM_STREAM_NOT_FOUND

[0076] Data types used herein are defined in Table 3:

Table 3

Symbol	Description
S8	Signed 8-bit integer.
U8	Unsigned 8-bit integer.
S16	Signed 16-bit integer.
U16	Unsigned 16-bit integer.
S32	Signed 32-bit integer.
U32	Unsigned 32-bit integer.
T_HostWord	A word on the Host processor.
T_DSP_Word	A word on the DSP processor.
BOOL	Boolean value (TRUE or FALSE).
T_HostPtr	Pointer on the Host processor.
T_DSP_Ptr	Pointer on the DSP processor.
T_DeviceID	Processor device ID.
T_TaskID	A structure containing fields for a device ID and a processor local task ID.
T_ChanID	A structure containing fields for a device ID and a processor local channel ID.
T_MsgID	Message ID.
T_DSP_ID	An object ID on the DSP.
T_Count	Data type for a count.
T_Size	Data type for a size.
T_HostCallBack	Value used when tasks send message back to the Host.
T_ReplyRef	M ssag r ply r ference.
T_LocalTaskID	Local task ID.
T_LocalChanID	Local chann l ID.

Table 3 (continued)

Symbol	Description
T_DSP_Name	Name for DSP objects (RTOS dependent).
T_CodeHdr	Code header structure for a DSP Library entry.
T_TaskCreate	Task creation structure.

[0077] These tables define the messages passing between devices (i.e. Host to DSP 16). The device IDs present as parameters in the corresponding function calls are not incorporated in the messages since they are used to actually route the message to the device. Similarly, task IDs that include a device ID as their upper half for the function call will not include the device ID in the message, but only the DSP's local task ID portion.

Table 4

DSP-API Messages			
Message	Send Parameters	Reply Parameters	Direction Host ⇔ DSP
GET_MIPS	None	U32 mips	→
GET_MEM_AVAIL		T_Size progmem T_Size datamem	→
ALLOC_MEM	U16 mempage T_Size size	T_DSP_Word *memptr U16 errcode	→
FREE_MEM	U16 mempage T_DSP_Word *memptr	U16 errcode	→
PUT_BLOB	T_DSP_Ptr destaddr U16 mempage T_Size size T_DSP_Word BLOB[size]	U16 errcode	→
CREATE_TASK	T_TaskCreate tcs	T_TaskID TaskID U16 errcode	→
START_TASK	T_TaskID TaskID	U16 errcode	→
SUSPEND_TASK	T_TaskID TaskID	U16 errcode	→
RESUME_TASK	T_TaskID TaskID	U16 errcode	→
DELETE_TASK	T_TaskID TaskID	U16 errcode	→
CHANGE_PRIORIT Y	T_TaskID TaskID U16 newpriority	U16 oldpriority U16 errcode	→
GET_TASK_STATUS	T_TaskID TaskID	U16 status U16 priority T_ChanID Input T_ChanID Output U16 errcode	→
GET_ID	T_DSP_Name Name	T_DSP_ID ID U16 errcode	→

Table 5

DSP Interfa e Layer / Channel Interface Lay r Messages			
Message	Send Parameters	Reply Parameters	Direction Host ⇔ DSP
CREATE_CHANNEL	T_Size framesize T_Count numframes	T_ChanID ChannelID U16 rrcode	→

Table 5 (continued)

DSP Interface Layer / Channel Interface Layer Messages			
M ssage	Send Parameters	Reply Parameters	Direction Host ⇔ DSP
DELETE_CHANNEL	T_ChanID ChannelID	U16 errcode	→
CREATE_STREAM	T_Size FIFOsize	T_StrmID StreamID U16 errcode	→
DELETE_STREAM	I_StrmID StreamID	U16 errcode	→

Table 6

Debug Messages			
Message	Send Parameters	Reply Parameters	Direction Host ⇔ DSP
READ_MEM	U16 mempage T_DSP_Ptr addr T_Count count	T_DSP_Word mem[count] U16 errcode	→
WRITE_MEM	U16 mempage T_DSP_Ptr addr	U16 errcode	→
	T_Count count T_DSP_Word mem[count]		
READ_REG	U16 RegID	DSP_WORD regvalue U16 errcode	→
WRITE_REG	U16 RegID T_DSP_Word regvalue	U16 errcode	→
SET_BREAK	T_DSP_Ptr addr	U16 errcode	→
CLR_BREAK	T_DSP_Ptr addr	U16 errcode	→
BREAK_HIT	T_DSP_Ptr addr	U16 ACK	←

[0078] Figures 4 - 6 illustrate an embodiment for downloading native code to a target processor (i.e., the host 12 or DSP 16) in a secure and efficient manner. This embodiment for downloading code could be used, for example, in downloading code from the Internet, or other global network, from a Local or Wide Area Network, or from a peripheral device, such as a PC Card or Smartcard.

[0079] In Figure 4, an embodiment of a JAVA Bean 90 is shown, where the Bean 90 acts as a wrapper for native code 92. The Bean further includes several attributes 94, listed as a Code Type attribute 94a, a Code Size attribute 94b and a MIPS Required attribute 94c. The Bean 90 has several actions 96, including a Load Code action 96a, a Load Parameters action 96b and an Execute Parameter 96c.

[0080] In operation, the Load Code action 96a is used to load external native code (native to the target processor) into the Bean. Since JAVA Beans have persistence, the Bean 90 can store its internal state, including the native code 92 and the attributes 94. The Load Parameters action 96b retrieves parameters from the native code 92 (using, for example, the COFF file format described above) and stores the parameters as attributes 94a-c. The Execute action 96c executes tasks installed in the DSP 16.

[0081] Figure 5 illustrates use of the Bean 90 to download code to the target processor. In this example, it is assumed that the target processor is the DSP 16 (or one of multiple DSPs 16), although it could be used to download native code to the host processor 12 as well. Further, it is assumed that the desired Bean 90 is resident in a network server, such as a LAN server or an Internet server, although the Bean could be resident in any device in communication with the platform 10, such as a Smartcard. For a wireless data platform 10, the connection to the network server 100 will often be wireless.

[0082] In Figure 5, the platform 10 is coupled to a network server 100. Th host processor 12, as shown in greater detail in Figure 2, may ex cute on or more JAVA applets 41 through a JAVA virtual machine 45. In ord r to download new code, the host 12 loads an applet 41 containing the Bean 90 from the network s rver 100 or the Bean, without the containing applet, can be downloaded from the s rver 100. Once the wrapper Bean 90 has been retrieved, it can be queried for the size of the native code, code type (for which processor is the code int nded) and MIPs r quired. If

the intended processor has sufficient resources to run the code 92, the code 92 can be installed to execute on the intended processor, either the host processor 12 or DSP 16 in the architecture shown in Figure 5. Typically, the native code 92 will be unlinked, compiled code. Thus, the cross linker 82 of the DSP-API 50 will link the code to an available memory location. The Bean would pass the binary native code 92 to the dynamic cross linker 82, which would install and execute the code.

[0083] A typical manner in which a download of native code might occur is when the user is running an applet 41 in which a DSP function is desired. First, the applet 41 would check to see if the desired code was installed as a task 60 in the DSP or was available in the DSP Library 58. If so, the task could be executed without a download.

[0084] If the task is not stored in the DSP 16 or the DSP library 58, an object (referred to as the " DSPLoader" object herein) could be created to load the Bean 90. If the DSPLoader class is local on the host 12, JAVA will check to see if the Bean is available locally as well. In a first instance, there may be a Bean with the code stored locally. If so, the code from the Bean is installed to the DSP 16 (or to whichever processor specified by the Code Type). If a Bean without the code is stored locally, the Bean can retrieve the code from the appropriate server.

[0085] On the other hand, if the DSPLoader object is not local, then JAVA will load the Bean 90 from the server that wrote the applet 41. The code from the Bean will then be installed as described above.

[0086] While the downloading of native code is described in connection with the use of a JAVA Bean, it could also be accomplished by wrapping the code within another language, such as an ActiveX applet.

[0087] Using a JAVA Bean (or other applet) as a wrapper to the native code has significant advantages. First, it allows a simple, standard method for loading code onto one of a plurality of processors. The Bean is created, code is loaded into the Bean and the code is linked to the appropriate processor. Without wrapping the code within the Bean, the process may take several hundred steps. Second, it allows multiple pieces of native code to be combined by a single applet, providing for complex applications to be generated from multiple discrete routines using a single applet to combine the routines as desired. Third, it takes advantage of the language's security features, thereby protecting not only the JAVA code in the Bean 90, but the native code 92 as well. Other languages, such as ActiveX, have security features as well.

[0088] Two of the most important security features are digital signing and encryption. A JAVA Bean or ActiveX applet may be signed by the source of the code; when the Bean or applet is downloaded, the signature is verified by the receiving application, which has a list of trusted sources. If the Bean or applet is signed by a trusted source, it can be decrypted using standard techniques. Accordingly, the native code is encrypted during transmission along with the code of the Bean or applet, preventing unauthorized modification of the code. Because the native code is secure and comes from a trusted source, the attributes can also be trusted as accurate.

[0089] Figure 6 illustrates a flow chart describing the process of downloading native code for a processor using a JAVA Bean, it being understood that the native code could be wrapped in an applet of a different language using similar techniques. In step 110, the encrypted, digitally signed Bean 90 is downloaded to a device running a JAVA virtual machine. In step 112, the signature is verified. If it is not from a source listed as a trusted source, exception processing is enabled in step 114. In the case of the Bean coming from a trusted source, the exception processing function may give the user an opportunity to accept the Bean, if the user is comfortable with the source. If the signature is invalid, the exception processing may delete the Bean 90 and send an appropriate error message to the user.

[0090] If the signature is valid and comes from a trusted source, the Bean is decrypted in step 116. This step decrypts both the JAVA code and the native code in the Bean. In step 118, the attributes are retrieved from the Bean 90 and in step 120 the applet determines whether the appropriate processor has sufficient resources to run the code. If not, th exception processing step 114 may decline to install the native code, or steps may be taken to free resources. If there are sufficient resources, the code is linked using the cross-linker and installed on the desired processor in step 122. In step 124, the native code is executed.

[0091] Sample JAVA script for a Bean 90 is provided hereinbelow:

```
package ti.dsp.loader;

import java.awt.*;
import java.io.*;
import java.net.*;

public class NativeBean extends Canvas implements Serializable
{
    public NativeBean() {

        setBackground(Color.white);

        funcData = new ByteArrayOutputStream();

        try {
            funcCodeBase = new URL("http://localhost");
        }
        catch (MalformedURLException e) {
```

```
    }
}

public Dimension getMinimumSize() {

    return new Dimension(50, 50);
}

public void loadCode() {

    URL baseURL = null;

    try {
        baseURL = new URL(funcCodeBase.toString() + "/" + myFunction);
    }
    catch (MalformedURLException e) {
    }

    DataInputStream source = null;
    int read;
    byte[] buffer;

    buffer = new byte[1024];
    try {
        source = new DataInputStream(baseURL.openStream());
    }
    catch (IOException e) {
        System.out.println("IOException creating streams: " + e);
    }

    codeSize = 0;

    funcData.reset();

    try {
        while (true) {

            read = source.read(buffer);

            if (read == -1)
                break;

            funcData.write(buffer, 0, read);
        }
    }
    catch (IOException e) {
        System.out.println("IOException: " + e);
    }
```

```
        codeSize = funcData.size();
        System.out.println("Code size = " + codeSize);

        try {
            source.close();
        }
        catch (IOException e) {
            System.out.println("IOException closing: " + e);
        }
    }

    public synchronized String getFunctionName() {

        return myFunction;
    }

    public void setFunctionName(String function) {

        myFunction = function;
    }

    public synchronized String getCodeBase() {

        return funcCodeBase.toString();
    }

    public void setCodeBase(String newBase) {

        try {
            funcCodeBase = new URL(newBase);
        }
        catch (MalformedURLException e) {
        }
    }

    public void installCode() {

        FileOutputStream destination = null;
        File libFile = new File(myFunction);

        try {
            destination = new FileOutputStream(libFile);
        }
        catch (IOException e) {
            System.out.println("IOException creating streams: " + e);
        }

        if (destination != null) {
```

```
                try {
                    funcData.writeTo(destination);
                }
                catch (IOException e) {
                    System.out.println("IO Exception installing native code: " + e);
                }
            }
        }
            linkCode(funcData)

        public void loadParameters() {
        }

        public void execute() {
        }

        public synchronized int getCodeSize() {

            return codeSize;
        }

        public synchronized int getCodeType() {

            return codeType;
        }

        public void setCodeType(int newType) {

            codeType = newType;
        }

        private int codeSize = 0;
        private int codeType = 1;
        private String myFunction = "";
        private URL funcCodeBase = null;
        private ByteArrayOutputStream funcData = null;
    }
```

[0092] In the script set forth above, the NativeBean() routine creates the Bean 90 which will hold the native code. The loadCode() routine gets the native code from the server. The getFunctionName() and getCodeBase() routines retrieve attributes. The installCode() routine calls the cross linker to link the native code to the DSP and to load the linked code. The loadParameters() routine instructs the Bean to examine the native code and determine its attribut s. The getCodesize() and getCodetype() routines transfer the attributes to the requesting applet.

[0093] Although the teachings disclosed herein have been directed to certain exemplary embodiments, various modifications of these embodiments, as well as alternative embodiments, will be suggested to those skilled in the art.

[0094] Further and particular embodiments of the invention will now be enumerated with reference to the following numbered clauses.

1. A mobil electronic device, comprising:

a coprocessor for xecuting nativ code;
a host proc ssor system operable to execute native code corresponding to the host processor syst m and processor independent code, said host processor system operable to dynamically change the tasks performed

by the digital signal coprocessor; and
circuitry for communicating between said host processor system and said coprocessor.

2. The mobile electronic device of clause 1 and further comprising network interface circuitry for receiving data from a network.

3. The mobile electronic device of clause 2 wherein said network interface circuitry comprises wireless network circuitry.

4. The mobile electronic device of clause 3 wherein said network interface circuitry comprises circuitry for interfacing with a global network.

5. A method of controlling a mobile electronic device comprising *the* steps of:

executing native code in a coprocessor;
executing both native code and processor independent code in a host processor system;
dynamically changing the tasks performed by the digital signal coprocessor with said host processor system; and
communicating between said host processor system and said coprocessor.

6. The method of clause 5 and further comprising the step of receiving code through a network interface.

7. The method of clause 6 and further comprising the step of receiving code through a wireless network interface.

8. The method of clause 6 or 7 and further comprising the step of receiving code through a wireless network interface from a global network.

9. A mobile electronic device, comprising:

a plurality of coprocessors;
a host processor system operable to:

execute source code;
identify one or more sections of source code to be executed on one or more of said coprocessors; and
for each identified section of source code, determining a corresponding coprocessor; and
for each identified section of source code, compile said identified section of code into the native code associated with said corresponding coprocessor and install said native code onto said corresponding coprocessor; and

circuitry for communicating between said host processor system and said coprocessors.

10. The mobile electronic device of clause 9 wherein one or more of said coprocessors comprise digital signal processors.

Claims

1. A mobile electronic device, comprising:

a coprocessor for executing native code;
a host processor operable to execute native code corresponding to the host processor and processor independent code, said host processor operable to dynamically change the tasks performed by the digital signal coprocessor; and
circuitry for communicating between said host processor and said coproc ssor.

2. Th mobile electronic d vice of Claim 1, wherein said coprocessor compris s a digital signal processor.

3. The mobile el ctronic device of Claim 1 or Claim 2, wherein said processor ind pendent code compris s JAVA.

4. The mobile electronic device of any preceding claim , wherein said host processor system is arranged to generate native code for said coprocessor.

5. The mobile electronic device of any preceding claim, wherein said host processor is arranged to generate native code for said coprocessor by compiling processor independent source code.

6. The mobile electronic device of any preceding claim, wherein said host processor is arranged to compile identified blocks of source code.

7. The mobile electronic device of any preceding claim. wherein said host processor system is arranged to identify blocks of source code that could be executed on the coprocessor and to compile said blocks of code.

8. The mobile electronic device of any preceding claims, further comprising:
a memory for storing a library of routines that can be downloaded to said coprocessor for execution.

9. The mobile electronic device of any preceding claim further comprising a hardware language accelerator.

10. The mobile electronic device of any preceding claim wherein said hardware accelerator comprises a JAVA accelerator.

11. The mobile electronic device of any preceding claim further comprising network interface circuitry for receiving data from a network.

12. A method of controlling a mobile electronic device comprising of:

executing native code in a coprocessor;
executing both native code and processor independent code in a host processor
dynamically changing the tasks performed by the digital signal coprocessor with said host processor and communicating between said host processor system and said coprocessor.

13. The method of claim 12 wherein said step of executing native code in a coprocessor comprises executing native code in a digital signal processor.

14. The method of claims 12 and 13 further comprising generating native code for coprocessor in said general processing system.

15. The method of claim 14 wherein said step of generating native code comprises the step of generating native code by compiling processor independent source code.

16. The method of any of claims 12 to 15 further comprising identifying blocks of said source code to compile for execution on said coprocessor.

17. The method of any of claims 12-16 further comprising storing a library of routines for downloading from said host processor system to said coprocessor for execution.

18. A mobile electronic device, comprising:

a plurality of coprocessors;
a host processor system operable to:

execute source code;
identify one or more portions of source code to be executed on one or more of said coprocessors; and
for each identified portion of source code, determining a corresponding coprocessor; and
for each id ntifi d portion of sourc cod , compil said id ntified portion of cod into th nativ code associated with said corresponding coprocessor and install said native code onto said corr sponding coprocessor; and

circuitry for communicating between said host processor syst m and said coprocessors.

19. A method of controlling a mobile electronic device, comprising:

executing source code on a host processor system;
identifying one or more portions of source code to be executed on one or more coprocessors: and
for each identified portion of source code, determining a corresponding coprocessor; and
for each identified portion of source code, compiling said identified portion of code into the native code associated with said corresponding coprocessor and installing said native code onto said corresponding coprocessor; and
communicating between said host processor system and said coprocessors.

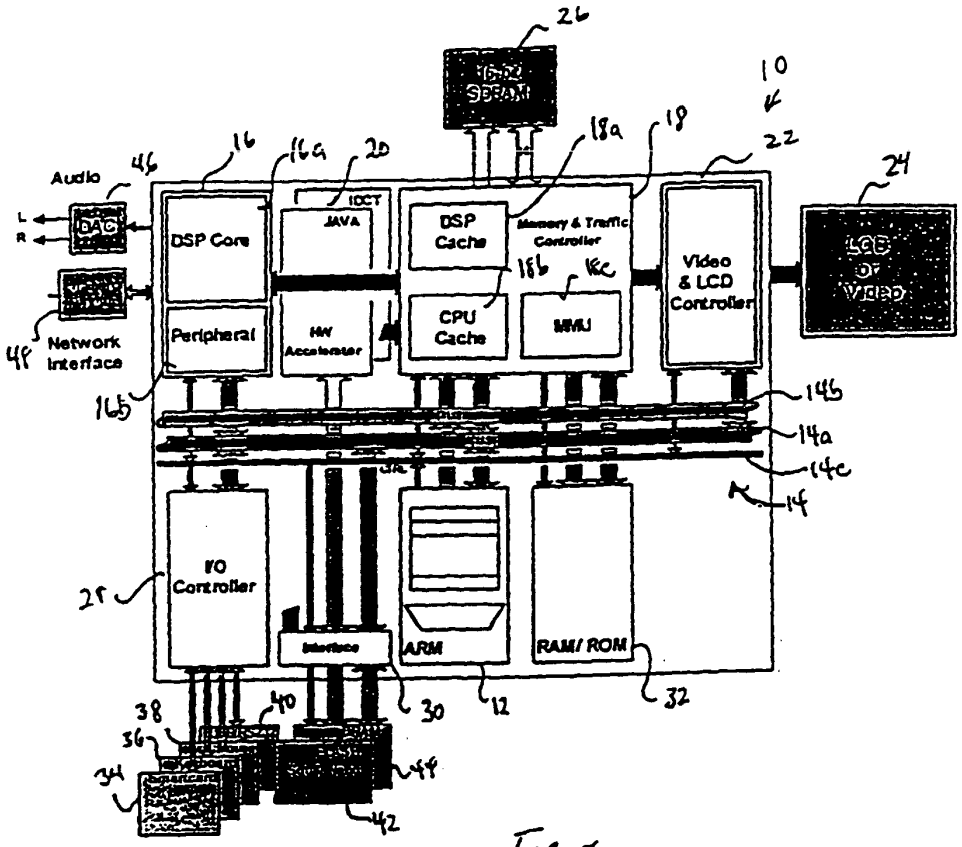

26
SDRAM
10
18a
18
22
24
Audio
46
16
16a
20
L
R
DAC
IOCT
JAVA
DSP Core
DSP Cache
Memory & Traffic Controller
Video & LCD Controller
LCD or Video
Network Interface
48
Peripheral
HW Accelerator
CPU Cache
MMU
18b
18c
16b
14b
14a
14c
14
I/O Controller
28
Interface
ARM
RAM/ ROM
30
12
32
38
40
36
34
44
42

FIG 1

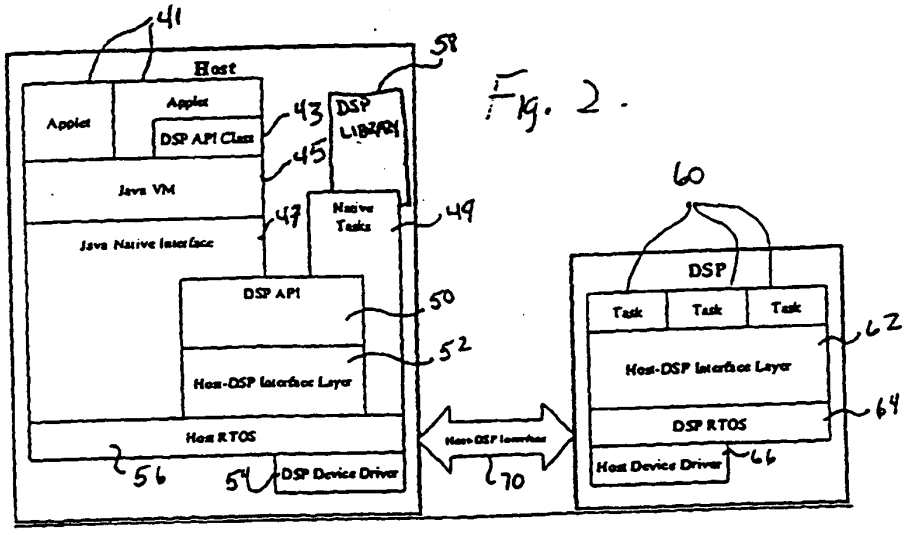

41
Host
58
Applet
Applet
DSP API Class
43
DSP LIBRARY
45
Java VM
47
Native Tasks
49
Java Native Interface
DSP API
50
52
Host-DSP Interface Layer
Host RTOS
56
54
DSP Device Driver
60
DSP
Task
Task
Task
62
Host-DSP Interface Layer
DSP RTOS
64
Host Device Driver
66
70

Fig. 2.

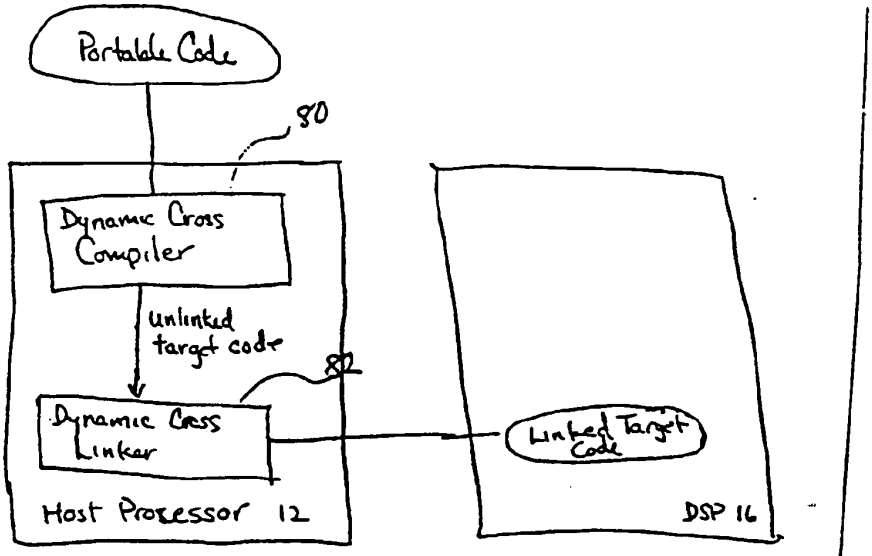


Figure 3

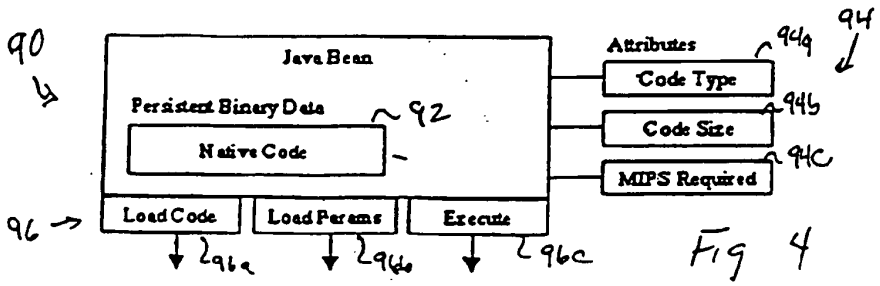


Fig 4

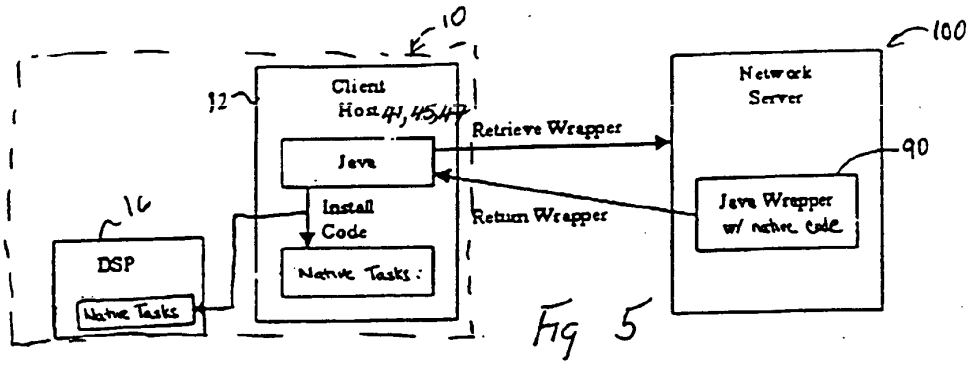

10
100
Client
Host 41, 45, 47
12
Java
Install
Code
Native Tasks
16
DSP
Native Tasks
Retrieve Wrapper
Return Wrapper
Network
Server
90
Java Wrapper
w/ native code

Fig 5

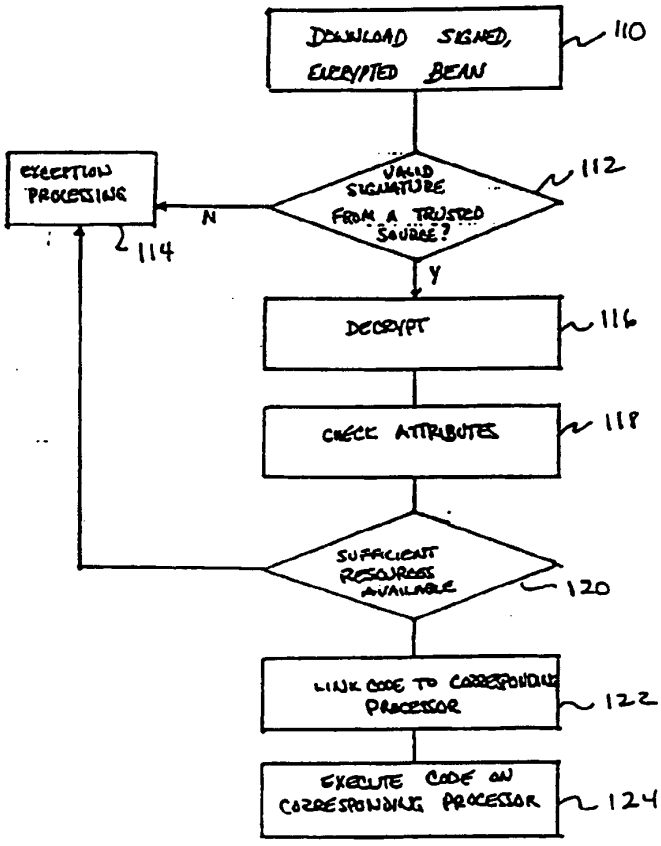

DOWNLOAD SIGNED, ENCRYPTED BEAN
110
VALID SIGNATURE FROM A TRUSTED SOURCE?
112
N
EXCEPTION PROCESSING
114
Y
DECRYPT
116
CHECK ATTRIBUTES
118
SUFFICIENT RESOURCES AVAILABLE
120
LINK CODE TO CORRESPONDING PROCESSOR
122
EXECUTE CODE ON CORRESPONDING PROCESSOR
124

FIG. 6

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 98 31 0312

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
P,X	WO 98 40978 A (SAGEM ;DEMEURE JEAN ANDRE (FR); DIMECH JEAN MARC (FR)) 17 September 1998 * page 4, line 22 - line 27 * * page 5, line 25 - line 28 * * page 8, line 26 - line 29 *	1,2,4, 11,12, 18,19	H04Q7/32 H04B1/38 G06F9/38
P,X	EP 0 869 691 A (DEUTSCHE TELEKOM AG) 7 October 1998 * column 2, line 4 - line 22 *	1-4, 11-14, 18,19	
A	GB 2 310 575 A (WESTINGHOUSE ELECTRIC CORP) 27 August 1997 * page 5, line 16 - line 25 *	1,2,12, 18,19	
A	WO 97 26750 A (CELLPORT LABS INC) 24 July 1997 * page 18, line 6 - page 22, line 26 *	1,12,18, 19	
A	US 4 862 407 A (FETTE BRUCE A ET AL) 29 August 1989 * column 4, line 49 - line 58 * * column 13, line 14 - line 18 *	1,12,18, 19	TECHNICAL FIELDS SEARCHED (Int.Cl.6) H04Q H04M G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
BERLIN	31 May 1999	Leouffre, M

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 98 31 0312

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

31-05-1999

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9840978 A	17-09-1998	FR 2760917 A	18-09-1998
		FR 2760918 A	18-09-1998
		AU 6921998 A	29-09-1998
EP 0869691 A	07-10-1998	DE 19713965 A	08-10-1998
GB 2310575 A	27-08-1997	AU 1264397 A	28-08-1997
WO 9726750 A	24-07-1997	US 5732074 A	24-03-1998
		AU 1525197 A	11-08-1997
		CA 2243454 A	24-07-1997
		EP 0875111 A	04-11-1998
US 4862407 A	29-08-1989	NONE	

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Problem Image Mailbox.

JONES DAY®

Facsimile Transmission

North Point, 901 Lakeside Avenue • Cleveland, Ohio 44114-1190 • (216) 586-3939
Facsimile: (216) 579-0212
dlpejeau@jonesday.com

August 21, 2003

Please hand deliver the following facsimile to:

Name: **Office of Initial Patent Examination's Filing Receipt Corrections**

Facsimile No.: **703-746-9195**

Company: **United States Patent & Trademark Office**

Number of pages (including this page): 3

Telephone No.:

From: **Debra L. Pejeau**

Title: **Patent & Trademark Assistant**

Send Copies To:

Direct Telephone No.: **(216) 586-7387**

JP No.: **JP259360**

☐ Copies distributed _____ Operator's Initials

CAM No.: 555255-012-423

Re:

Originals Will Not Follow

Message:

Application Number 10/381,219 (Int'l Filing Date 09/20/2001)

Dear Sir or Madam,

Please correct the application title on the attached Filing Receipt as indicated and issue a Corrected Filing Receipt. Thank you.

Respectfully submitted,
Debra L. Pejeau

Please call us immediately if the facsimile you receive is incomplete or illegible. Please ask for the facsimile operator.

CLI-1052528v1

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING DATE	GRP ART UNIT	FIL FEE REC'D	ATTY.DOCKET.NO	DRAWINGS	TOT CLAIMS	IND CLAIMS
10/381,219	03/20/2003	2131	3258	555255012423	7	109	12

CONFIRMATION NO. 9761

David B Cochran
Jones Day
North Point
901 Lakeside Avenue
Cleveland, OH 44114-1190

FILING RECEIPT

OC000000010312502

Date Mailed: 06/25/2003

Receipt is acknowledged of this regular Patent Application. It will be considered in its order and you will be notified as to the results of the examination. Be sure to provide the U.S. APPLICATION NUMBER, FILING DATE, NAME OF APPLICANT, and TITLE OF INVENTION when inquiring about this application. Fees transmitted by check or draft are subject to collection. **Please verify the accuracy of the data presented on this receipt. If an error is noted on this Filing Receipt, please write to the Office of Initial Patent Examination's Filing Receipt Corrections, facsimile number 703-746-9195. Please provide a copy of this Filing Receipt with the changes noted thereon. If you received a "Notice to File Missing Parts" for this application, please submit any corrections to this Filing Receipt with your reply to the Notice. When the USPTO processes the reply to the Notice, the USPTO will generate another Filing Receipt incorporating the requested corrections (if appropriate).**

Applicant(s)

David P Yach, Waterloo, ON, CANADA;
Michael S Brown, Waterloo, ON, CANADA;
Herbert A Little, Waterloo, ON, CANADA;

Domestic Priority data as claimed by applicant

This application is a 371 of PCT/CA01/01344 09/20/2001
which claims benefit of 60/234,152 09/21/2000
and claims benefit of 60/235,354 09/26/2000
and claims benefit of 60/270,663 02/20/2001

Foreign Applications

Projected Publication Date: 09/25/2003

Non-Publication Request: No

Early Publication Request: No

Title Software
Code signing system and method

Preliminary Class
713

LICENSE FOR FOREIGN FILING UNDER
Title 35, United States Code, Section 184
Title 37, Code of Federal Regulations, 5.11 & 5.15

GRANTED

The applicant has been granted a license under 35 U.S.C. 184, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" followed by a date appears on this form. Such licenses are issued in all applications where the conditions for issuance of a license have been met, regardless of whether or not a license may be required as set forth in 37 CFR 5.15. The scope and limitations of this license are set forth in 37 CFR 5.15(a) unless an earlier license has been issued under 37 CFR 5.15(b). The license is subject to revocation upon written notification. The date indicated is the effective date of the license, unless an earlier license of similar scope has been granted under 37 CFR 5.13 or 5.14.

This license is to be retained by the licensee and may be used at any time on or after the effective date thereof unless it is revoked. This license is automatically transferred to any related applications(s) filed under 37 CFR 1.53(d). This license is not retroactive.

Th grant of a license does not in any way lessen the responsibility of a licensee for the security of the subject matter as imposed by any Government contract or the provisions of existing laws relating to espionage and the national security or the export of technical data. Licensees should apprise themselves of current regulations especially with respect to certain countries, of other agencies, particularly the Office of Defense Trade Controls, Department of State (with respect to Arms, Munitions and Implements of War (22 CFR 121-128)); the Office of Export Administration, Department of Commerce (15 CFR 370.10 (j)); the Office of Foreign Assets Control, Department of Treasury (31 CFR Parts 500+) and the Department of Energy.

NOT GRANTED

No license under 35 U.S.C. 184 has been granted at this time, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" DOES NOT appear on this form. Applicant may still petition for a license under 37 CFR 5.12, if a license is desired before the expiration of 6 months from the filing date of the application. If 6 months has lapsed from the filing date of this application and the licensee has not received any indication of a secrecy order under 35 U.S.C. 181, the licensee may foreign file the application pursuant to 37 CFR 5.15(b).

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

U.S. APPLICATION NUMBER NO	FIRST NAMED APPLICANT	ATTY. DOCKET NO.
10/381,219	David P Yach	555255012423

INTERNATIONAL APPLICATION NO.	
PCT/CA01/01344	
I.A. FILING DATE	PRIORITY DATE
09/20/2001	09/21/2000

David B Cochran
Jones Day
North Point
901 Lakeside Avenue
Cleveland, OH 44114-1190

CONFIRMATION NO. 9761

371 ACCEPTANCE LETTER

OC000000010312504

Date Mailed: 06/25/2003

NOTICE OF ACCEPTANCE OF APPLICATION UNDER 35 U.S.C 371 AND 37 CFR 1.495

The applicant is hereby advised that the United States Patent and Trademark Office in its capacity as a Designated / Elected Office (37 CFR 1.495), has determined that the above identified international application has met the requirements of 35 U.S.C. 371, and is ACCEPTED for national patentability examination in the United States Patent and Trademark Office.

The United States Application Number assigned to the application is shown above and the relevant dates are:

03/20/2003	03/20/2003
DATE OF RECEIPT OF 35 U.S.C. 371(c)(1), (c)(2) and (c)(4) REQUIREMENTS	DATE OF RECEIPT OF ALL 35 U.S.C. 371 REQUIREMENTS

A Filing Receipt (PTO-103X) will be issued for the present application in due course. **THE DATE APPEARING ON THE FILING RECEIPT AS THE " FILING DATE" IS THE DATE ON WHICH THE LAST OF THE 35 U.S.C. 371 REQUIREMENTS HAS BEEN RECEIVED IN THE OFFICE. THIS DATE IS SHOWN ABOVE.** *The filing date of the above identified application is the international filing date of the international application (Article 11(3) and 35 U.S.C. 363).* Once the Filing Receipt has been received, send all correspondence to the Group Art Unit designated thereon.

The following items have been received:

- Copy of the International Application filed on 03/20/2003
- Copy of the International Search Report filed on 03/20/2003
- Copy of IPE Report filed on 03/20/2003
- Preliminary Amendments filed on 03/20/2003
- Oath or Declaration filed on 03/20/2003
- Request for Immediate Examination filed on 03/20/2003
- Copy of references cited in ISR filed on 03/20/2003
- U.S. Basic National Fees filed on 03/20/2003
- Assignee Statement filed on 03/20/2003

Applicant is reminded that any communications to the United States Patent and Trademark Office must be mailed to the address given in the heading and include the U.S. application no. shown above (37 CFR 1.5)

TAMALA D HOLLAND
Telephone: (703) 305-5483

PART 3 - OFFICE COPY

FORM PCT/DO/EO/903 (371 Acceptance Notice)

FORM PTO-1390 (REV. 01-2003) U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FILING UNDER 35 U.S.C. 371

ATTORNEY'S DOCKET NUMBER: 555255012423

U.S. APPLICATION NO. (If known, see 37 CFR 1.5): 10/381219

INTERNATIONAL APPLICATION NO.	INTERNATIONAL FILING DATE	PRIORITY DATE CLAIMED
PCT/CA01/01344	September 20, 2001	September 21, 2000

TITLE OF INVENTION: SOFTWARE CODE SIGNING SYSTEM AND METHOD

APPLICANT(S) FOR DO/EO/US: David P. Yach; Michael S. Brown; Herbert A. Little

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. ☒ This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371.
2. ☐ This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.
3. ☒ This is an express request to begin national examination procedures (35 U.S.C. 371(f)). The submission must include items (5), (6), (9) and (21) indicated below.
4. ☐ The US has been elected (Article 31).
5. ☒ A copy of the International Application as filed (35 U.S.C. 371(c)(2))
   a. ☒ is attached hereto (required only if not communicated by the International Bureau).
   b. ☐ has been communicated by the International Bureau.
   c. ☐ is not required, as the application was filed in the United States Receiving Office (RO/US).

N/A

6. ☐ An English language translation of the International Application as filed (35 U.S.C. 371(c)(2)).
   a. ☐ is attached hereto.
   b. ☐ has been previously submitted under 35 U.S.C. 154(d)(4).
7. ☐ Amendments to the claims of the International Application under PCT Article 19 (35 U.S.C. 371(c)(3))
   a. ☐ are attached hereto (required only if not communicated by the International Bureau).
   b. ☐ have been communicated by the International Bureau.
   c. ☐ have not been made; however, the time limit for making such amendments has NOT expired.
   d. ☐ have not been made and will not be made.
8. ☐ An English language translation of the amendments to the claims under PCT Article 19 (35 U.S.C. 371 (c)(3)).
9. ☒ An oath or declaration of the inventor(s) (35 U.S.C. 371(c)(4)). (3 pgs)
10. ☐ An English language translation of the annexes of the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371(c)(5)).

Items 11 to 20 below concern document(s) or information included:

11. ☐ An Information Disclosure Statement under 37 CFR 1.97 and 1.98.
12. ☒ An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included. (5 pgs)
13. ☒ A preliminary amendment. (21 pgs)
14. ☐ An Application Data Sheet under 37 CFR 1.76.
15. ☐ A substitute specification.
16. ☒ A power of attorney ~~and/or change of address letter~~. (2 pgs)
17. ☐ A computer-readable form of the sequence listing in accordance with PCT Rule 13*ter*.2 and 37 CFR 1.821 - 1.825.
18. ☐ A second copy of the published international application under 35 U.S.C. 154(d)(4).
19. ☐ A second copy of the English language translation of the international application under 35 U.S.C. 154(d)(4).
20. ☐ Other items or information:

DT09 Rec'd PCT/PTO 20 MAR 2003

U.S. APPLICATION NO. (if known, see 37 CFR 1.5)	INTERNATIONAL APPLICATION NO.	ATTORNEY'S DOCKET NUMBER
10/381219	PCT/CA01/01344	555255012423

21. ☒ The following fees are submitted:	CALCULATIONS PTO USE ONLY	
BASIC NATIONAL FEE (37 CFR 1.492 (a) (1) - (5)):		
Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO and International Search Report not prepared by the EPO or JPO **$1060.00**		
International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO **$900.00**		
International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO **$750.00**		
International preliminary examination fee (37 CFR 1.482) paid to USPTO but all claims did not satisfy provisions of PCT Article 33(1)-(4) **$720.00**		
International preliminary examination fee (37 CFR 1.482) paid to USPTO and all claims satisfied provisions of PCT Article 33(1)-(4) **$100.00**		
ENTER APPROPRIATE BASIC FEE AMOUNT =	$ 900	
Surcharge of **$130.00** for furnishing the oath or declaration later than 30 months from the earliest claimed priority date (37 CFR 1.492(e)).	$	

CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE		
Total claims	109 - 20 =	89	x **$18.00**	$ 1602	
Independent claims	12 - 3 =	9	x **$84.00**	$ 756	
MULTIPLE DEPENDENT CLAIM(S) (if applicable)			+ **$280.00**	$	
TOTAL OF ABOVE CALCULATIONS =				$ 2358	
☐ Applicant claims small entity status. See 37 CFR 1.27. The fees indicated above are reduced by 1/2. +				$	
SUBTOTAL =				$ 3258	
Processing fee of **$130.00** for furnishing the English translation later than 30 months from the earliest claimed priority date (37 CFR 1.492(f)).				$	
TOTAL NATIONAL FEE =				$ 3258	
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31). **$40.00** per property +				$ 40	
TOTAL FEES ENCLOSED =				$ 3298	
				Amount to be refunded:	$
				charged:	$

a. ☐ A check in the amount of $ _____ to cover the above fees is enclosed.

b. ☒ Please charge my Deposit Account No. 501432 in the amount of $ 3298 to cover the above fees. A duplicate copy of this sheet is enclosed. **(ref. 555255012423)

c. ☒ The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. 501432. A duplicate copy of this sheet is enclosed.

d. ☐ Fees are to be charged to a credit card. **WARNING:** Information on this form may become public. **Credit card information should not be included on this form.** Provide credit card information and authorization on PTO-2038.

NOTE: Where an appropriate time limit under 37 CFR 1.495 has not been met, a petition to revive (37 CFR 1.137 (a) or (b)) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO:
David B. Cochran, Esq.
Jones Day
901 Lakeside Ave./North Point
Cleveland, Ohio 44114

David Cochran
SIGNATURE

David B. Cochran
NAME

39,142
REGISTRATION NUMBER

FORM PTO-1390 (REV 01-2003) page 2 of 2

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In the application of	:	David P. Yach; Michael S. Brown; Herbert A. Little
Internat'l. Appl'n. No.	:	PCT/CA01/01344
Internat'l. Filing Date	:	09/20/2001
U.S. Serial No.	:	Not yet assigned
U.S. Filing Date	:	03/20/2003
Priority Date Claimed:		09/21/2000
Title	:	Software Code Signing System And Method
Art Unit	:	Not yet assigned
Examiner	:	Not yet assigned
Docket No.	:	555255012243

Date: March 20, 2003

Commissioner for Patents
Washington, D.C. 20231

Preliminary Amendment

Prior to taking up this case for initial examination, please amend the application as follows.

The Claims

Please cancel original claims 1-56.

Please add the following new claims 1-109.

1. (New) A code signing system for operation in conjunction with a software application having a digital signature and a signature identification, where the digital signature is associated with the signature identification, comprising:

an application platform;

an application programming interface (API) having an associated signature identifier, the API is configured to link the software application with the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application where the signature identifier corresponds to the signature identification.

2. (New) The code signing system of claim 1, wherein the virtual machine denies the software application access to the API if the digital signature is not authenticated.

3. (New) The code signing system of claim 1, wherein the virtual machine purges the software application if the digital signature is not authenticated.

4. (New) The code signing system of claim 1, wherein the code signing system is installed on a mobile device.

5. (New) The code signing system of claim 1, wherein the digital signature is generated by a code signing authority.

6. (New) A code signing system for operation in conjunction with a software application having a digital signature and a signature identification where the digital signature is associated with the signature identification, comprising:

an application platform;

a plurality of application programming interfaces (APIs) associated with a signature identifier, each configured to link the software application with a resource on the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the APIs by the software application where the signature identification corresponds to the signature identifier,

wherein the virtual machine verifies the authenticity of the digital signature in order to control access to the plurality of APIs by the software application.

7. (New) The code signing system of claim 6, wherein the plurality of APIs are included in an API library.

8. (New) The code signing system of claim 6, wherein one or more of the plurality of APIs is classified as sensitive and having an associated signature identifier, and wherein the virtual machine uses the digital signature and the signature identification to control access to the sensitive APIs.

9. (New) The code signing system of claim 8, wherein the code signing system operates in conjunction with a plurality of software applications, wherein one or more of the plurality of software applications has a digital signature and a signature identification, and wherein the virtual machine verifies the authenticity of the digital signature of each of the one or more of the plurality of software applications, where the signature identification corresponds to the signature identifier of the respective sensitive APIs, in order to control access to the sensitive APIs by each of the plurality of software applications.

10. (New) The code signing system of claim 6, wherein the resource on the application platform comprises a wireless communication system.

11. (New) The code signing system of claim 6, wherein the resource on the application platform comprises a cryptographic module which implements cryptographic algorithms.

12. (New) The code signing system of claim 6, wherein the resource on the application platform comprises a data store.

13. (New) The code signing system of claim 6, wherein the resource on the application platform comprises a user interface (UI).

14. (New) The code signing system of claim 1, further comprising:

a plurality of API libraries, each of the plurality of API libraries includes a plurality of APIs, wherein the virtual machine controls access to the plurality of API libraries by the software application.

15. (New) The code signing system of claim 14, wherein at least one of the plurality of API libraries is classified as sensitive;

wherein access to a sensitive API library requires a digital signature associated with a signature identification where the signature identification corresponds to a signature identifier associated with the sensitive API library;

wherein the software application includes at least one digital signature and at least one associated signature identification for accessing sensitive API libraries; and

wherein the virtual machine authenticates the software application for accessing the sensitive API library by verifying the one digital signature included in the software application that has a signature identification corresponding to the signature identifier of the sensitive API library.

16. (New) The code signing system of claim 1, wherein the digital signature is generated using a private signature key, and the virtual machine uses a public signature key to verify the authenticity of the digital signature.

17. (New) The code signing system of claim 16, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application; and

the virtual machine verifies the authenticity of the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and comparing the generated hash with the recovered hash.

18. (New) The code signing system of claim 4, wherein the API further comprises:

a description string that is displayed by the mobile device when the software application attempts to access the API.

19. (New) The code signing system of claim 1, wherein the application platform comprises an operating system.

20. (New) The code signing system of claim 1, wherein the application platform comprises one or more core functions of a mobile device.

21. (New) The code signing system of claim 1, wherein the application platform comprises hardware on a mobile device.

22. (New) The code signing system of claim 21, wherein the hardware comprises a subscriber identity module (SIM) card.

23. (New) The code signing system of claim 1, wherein the software application is a Java application for a mobile device.

24. (New) The code signing system of claim 1, wherein the API interfaces with a cryptographic routine on the application platform.

25. (New) The code signing system of claim 1, wherein the API interfaces with a proprietary data model on the application platform.

26. (New) The code signing system of claim 1, wherein the virtual machine is a Java virtual machine installed on a mobile device.

27. (New) A method of controlling access to sensitive application programming interfaces on a mobile device, comprising the steps of:

loading a software application on the mobile device that requires access to a sensitive application programming interface (API) having a signature identifier;

determining whether the software application includes a digital signature and a signature identification; and

denying the software application access to the sensitive API where the signature identification does not correspond with the signature identifier.

28. (New) The method of claim 27, comprising the additional step of:

purging the software application from the mobile device where the signature identification does not correspond with the signature identifier.

29. (New) The method of claim 27, wherein the digital signature and the signature identification are generated by a code signing authority.

30. (New) The method of claim 27, comprising the additional steps of:

verifying the authenticity of the digital signature where the signature identification corresponds with the signature identifier.; and

denying the software application access to the sensitive API where the digital signature is not authenticated.

31. (New) The method of claim 30, comprising the additional step of:

purging the software application from the mobile device where the digital signature is not authenticated.

32. (New) The method of claim 30, wherein the digital signature is generated by applying a private signature key to a hash of the software application, and wherein the step of verifying the authenticity of the digital signature is performed by a method comprising the steps of:

storing a public signature key that corresponds to the private signature key on the mobile device;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

33. (New) The method of claim 32, wherein the digital signature is generated by calculating a hash of the software application and applying the private signature key.

34. (New) The method of claim 27, comprising the additional step of:

displaying a description string that notifies a user of the mobile device that the software application requires access to the sensitive API.

35. (New) The method of claim 34, comprising the additional step of:

receiving a command from the user granting or denying the software application access to the sensitive API.

36. (New) A method of controlling access to an application programming interface (API) having a signature identifier on a mobile device by a software application created by a software developer, comprising the steps of:

receiving the software application from the software developer;

determining whether the software application satisfies at least one criterion;

appending a digital signature and a signature identification to the software application where the software application satisfies at least one criterion;;

verifying the authenticity of the digital signature appended to the software application where the signature identification corresponds with the signature identifier; and

providing access to the API to software applications where the digital signature is authenticated.

37. (New) The method of claim 36, wherein the step of determining whether the software application satisfies at least one criterion is performed by a code signing authority.

38. (New) The method of claim 36, wherein the step of appending the digital signature and the signature identification to the software application includes generating the digital signature comprising the steps of:

calculating a hash of the software application; and

applying a signature key to the hash of the software application to generate the digital signature.

39. (New) The method of claim 38, wherein the hash of the software application is calculated using the Secure Hash Algorithm (SHA1).

40. (New) The method of claim 38, wherein the step of verifying the authenticity of the digital signature comprises the steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash; and

authenticating the digital signature by comparing the calculated hash with the recovered hash.

41. (New) The method of claim 40, comprising the further step of denying the software application access to the API where the digital signature is not authenticated.

42. (New) The method of claim 40, wherein the signature key is a private signature key and the corresponding signature key is a public signature key.

43. (New) A method of controlling access to a sensitive application programming interface (API) having a signature identifier on a mobile device, comprising the steps of:

registering one or more software developers that are trusted to develop software applications which access the sensitive API;

receiving a hash of a software application;

determining whether the hash was sent by a registered software developer; and

generating a digital signature using the hash of the software application and a signature identification corresponding to the signature identifier where the hash was sent by the registered software developer;

wherein

the digital signature and the signature identification are appended to the software application; and

the mobile device verifies the authenticity of the digital signature in order to control access to the sensitive API by the software application where the signature identification corresponds with the signature identifier.

44. (New) The method of claim 43, wherein the step of generating the digital signature is performed by a code signing authority.

45. (New) The method of claim 43, wherein the step of generating the digital signature is performed by applying a signature key to the hash of the software application.

46. (New) The method of claim 45, wherein the mobile device verifies the authenticity of the digital signature by performing the additional steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash;

determining whether the digital signature is authentic by comparing the calculated hash with the recovered hash; and

denying the software application access to the sensitive API where the digital signature is not authenticated.

47. (New) A method of restricting access to application programming interfaces on a mobile device, comprising the steps of:

loading a software application having a digital signature and a signature identification on the mobile device that requires access to one or more application programming interfaces (APIs) having at least one signature identifier;

authenticating the digital signature where the signature identification corresponds with the signature identifier; and

denying the software application access to the one or more APIs where the software application does not include an authentic digital signature .

48. (New) The method of claim 47, wherein the digital signature and signature identification are associated with a type of mobile device.

49. (New) The method of claim 47, comprising the additional step of:

purging the software application from the mobile device where the software application does not include an authentic digital signature. .

50. (New) The method of claim 47, wherein:

the software application includes a plurality of digital signatures and signature identifications; and

the plurality of digital signatures and signature identifications includes digital signatures and signature identifications respectively associated with different types of mobile devices.

51. (New) The method of claim 50, wherein each of the plurality of digital signatures and associated signature identifications are generated by a respective corresponding code signing authority.

52. (New) The method of claim 47, wherein the step of determining whether the software application includes an authentic digital signature comprises the additional steps of:

verifying the authenticity of the digital signature where the signature identification corresponds with respective ones of the at least one signature identifier.

53. (New) The method of claim 51, wherein each of the plurality of digital signatures and signature identifications are generated by its corresponding code signing authority by applying a respective private signature key associated with the code signing authority to a hash of the software application.

54. (New) The method of claim 47, wherein the step of authenticating the digital signature where the signature identification corresponds with the signature identifier comprises the steps of:

verifying that the signature identification corresponds with the signature identifier authenticating the digital signature where signature identification corresponds with the signature identifier comprising the steps of:

storing a public signature key on a mobile device that corresponds to the private signature key associated with the code signing authority which generates the digital signature;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

55. (New) The method of claim 47, wherein:

the mobile device includes a plurality of APIs;

at least one of the plurality of APIs is classified as sensitive;

access to any of the plurality of APIs requires an authentic global signature;

access to each of the plurality of sensitive APIs requires an authentic global signature and an authentic digital signature associated with a signature identification;

the step of determining whether the software application includes an authentic digital signature and signature identification comprises the steps of:

determining whether the one or more APIs to which the software application requires access includes a sensitive API;

determining whether the software application includes an authentic global signature; and

determining whether the software application includes an authentic digital signature and signature identification where the one or more APIs to which the software application requires access includes a sensitive API and the software application includes an authentic global signature; and

the step of denying the software application access to the one or more APIs comprises the steps of:

denying the software application access to the one or more APIs where the software application does not include an authentic global signature; and

denying the software application access to the sensitive API where the one or more APIs to which the software application requires access includes a sensitive API, the software application includes an authentic global signature, and the software application does not include an authentic digital signature and signature identifier required to access the sensitive API.

56. (New) A code signing system for controlling access to application programming interfaces (APIs) having signature identificaters by software applications, the code signing system comprising:

a verification system for authenticating digital signatures provided by the respective software applications to access the APIs where the signature identifications correspond with the signature identificaters of the respective APIs and where a digital signature for a software application is generated with a signature identification corresponding to a signature identificater to access at least one API; and

a control system for allowing access to at least one of the APIs where the digital signature provided by the software application is authenticated by the verification system.

57. (New) The code signing system of claim 56, wherein a virtual machine comprises the verification system and the control system.

58. (New) The code signing system of claim 57, wherein the virtual machine is a Java virtual machine installed on a mobile device.

59. (New) The code signing system of claim 56, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

60. (New) The code signing system of claim 56, wherein the code signing system is installed on a mobile device and the software application is a Java application for a mobile device.

61. (New) The code signing system of claim 56, wherein the digital signature and the signature identification of the software application are generated by a code signing authority.

62. (New) The code signing system of claim 56, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

63. (New) The code signing system of claim 56, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and the verification system uses a public signature key to authenticate the digital signature.

64. (New) The code signing system of claim 63, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

65. (New) The code signing system of claim 56, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

66. (New) The code signing system of claim 56, wherein the APIs provides access to at least one of one or more core functions of a mobile device, an operating system, and hardware on a mobile device.

67. (New) The code signing system of claim 56, wherein verification of a global digital signature provided by the software application is required for accessing any of the APIs.

68. (New) A method of controlling access to application programming interfaces (APIs) having signature identifiers by software applications, the method comprising:

authenticating digital signatures provided by the respective software applications to access the APIs where the signature identifications correspond with the signature identifiers of the respective APIs and where a digital signature for a software application is generated with a signature identification corresponding to a signature identifier to access at least one API; and

allowing access to at least one of the APIs where the digital signature provided by the software application is authenticated.

69. (New) The method of claim 68, wherein one digital signature and one signature identification are provided by the software application access a library of at least one of the APIs.

70. (New) The method of claim 68, wherein the digital signature and the signature identification of the software application are generated by a code signing authority.

71. (New) The method of claim 68, wherein the APIs access at least one of a cryptographic module that implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

72. (New) The method of claim 68, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and a public signature key is used to authenticate the digital signature.

73. (New) The method of claim 72, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the digital signature is authenticated by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

74. (New) The method of claim 68, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

75. (New) The method of claim 68, wherein the APIs provides access to at least one of one or more core functions of a mobile device, an operating system, and hardware on a mobile device.

76. (New) The method of claim 68, wherein verification of a global digital signature provided by the software application is required for accessing any of the APIs

77. (New) A management system for controlling access by software applications to application programming interfaces (APIs) having at least one signature identifier on a subset of a plurality of mobile devices, the management system comprising:

a code signing authority for providing digital signatures and signature identifications to software applications that require access to at least one of the APIs with a signature identifier on the subset of the plurality of mobile devices, where a digital signature for a software application is generated with a signature identification corresponding to a signature identifier, and the signature identifications provided to the software applications comprise those signature identifications that correspond to the signature identifiers that are substantially only on the subset of the plurality of mobile devices; wherein each mobile device of the subset of the plurality of mobile devices comprises

a verification system for authenticating digital signatures provided by the respective software applications to access respective APIs where the digital identifications correspond to the digital identifiers of the respective APIs; and

a control system for allowing the respective software applications to access at least one of the APIs where the digital signatures provided by the respective software applications are authenticated by the verification system.

78. (New) The management system of claim 77, wherein a virtual machine comprises the verification system and the control system.

79. (New) The management system of claim 78, wherein the virtual machine is a Java virtual machine and the software applications are Java applications.

80. (New) The management system of claim 77, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

81. (New) The management system of claim 77, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

82. (New) The management system of claim 77, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and the verification system uses a public signature key to authenticate the digital signature.

83. (New) The management system of claim 82, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

84. (New) The management system of claim 77, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

85. (New) The management system of claim 77, wherein the subset of the plurality of mobile devices comprises mobile devices under the control of at least one of a corporation and a carrier.

86. (New) The management system of claim 77, wherein a global digital signature provided by the software application has to be authenticated before the software application is allowed access to any of the APIs on a mobile device of the subset of the plurality of mobile devices.

87. (New) A method of controlling access by software applications to application programming interfaces (APIs) having at least one signature identifier on a subset of a plurality of mobile devices, the method comprising:

generating digital signatures for software applications with signature identifications corresponding to respective signature identifiers of the APIs; and

providing the digital signatures and the signature identifications to software applications that require access to at least one of the APIs on the subset of the plurality of mobile devices, where the signature identifications provided to the software applications comprise those signature identifications that correspond to the signature identifiers that are substantially only on the subset of the plurality of mobile devices; wherein each mobile device of the subset of the plurality of mobile devices comprises

a verification system for authenticating digital signatures provided by the respective software applications to access respective APIs where the digital identifications correspond to the digital identifiers of the respective APIs; and

a control system for allowing the software application to access at least one of the APIs where the digital signature provided by the software application is authenticated by the verification system.

88. (New) The method of claim 87, wherein a virtual machine comprises the verification system and the control system.

89. (New) The method of claim 88, wherein the virtual machine is a Java virtual machine and the software applications are Java applications.

90. (New) The method of claim 87, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

91. (New) The method of claim 87, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

92. (New) The method of claim 87, wherein at least one of the digital signatures is generated using a private signature key under a signature scheme associated with a signature identification, and the verification system uses a public signature keys to authenticate said at least one of the digital signatures.

93. (New) The method of claim 92, wherein:

at least one of the digital signatures is generated by applying the private signature key to a hash of a software application under the signature scheme; and

the verification system authenticates said at least one of the digital signatures by generating a hash of the software application to obtain a generated hash, applying the public signature key to said at least one of the digital signatures to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

94. (New) The method of claim 87, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

95. (New) The method of claim 87, wherein the subset of the plurality of mobile devices comprises mobile devices under the control of at least one of a corporation and a carrier.

96. (New) A mobile device for a subset of a plurality of mobile devices, the mobile device comprising:

an application platform having application programming interfaces (APIs);

a verification system for authenticating digital signatures and signature identifications provided by the respective software applications to access the APIs; and

a control system for allowing a software application to access at least one of the APIs where a digital signature provided by the software application is authenticated by the verification system;

wherein a code signing authority provides digital signatures and signature identifications to software applications that require access to at least one of the APIs such that the digital signature for the software application is generated according to a signature scheme of a signature identification, and wherein the signature identifications provided to the software applications comprise those signature identifications that are substantially only authorized to allow access on the subset of the plurality of mobile devices.

97. (New) The mobile device of claim 96, wherein a virtual machine comprises the verification system and the control system.

98. (New) The mobile device of claim 97, wherein the virtual machine is a Java virtual machine and the software application is a Java application.

99. (New) The mobile device of claim 96, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

100. (New) The mobile device of claim 96, wherein the APIs of the application platform access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

101. (New) The mobile device of claim 96, wherein the digital signature is generated using a private signature key under the signature scheme, and the verification system uses a public signature key to authenticate the digital signature.

102. (New) The mobile device of claim 101, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

103. (New) The mobile device of claim 96, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

104. (New) A method of controlling access to application programming interfaces (APIs) of an application platform of a mobile device for a subset of a plurality of mobile devices, the method comprising:

receiving digital signatures and signature identifications from software applications that require to access the APIs

authenticating the digital signatures and the signature identifications; and

allowing a software application to access at least one of the APIs where a digital signature provided by the software application is authenticated;

wherein a code signing authority provides the digital signatures and the signature identifications to the software applications that require access to at least one of the APIs such that the digital signature for the software application is generated according to a signature scheme of a signature identification, and wherein the signature identifications provided to the software applications comprise those signature identifications that are substantially only authorized to allow access on the subset of the plurality of mobile devices.

105. (New) The method of claim 104, wherein one digital signature and one signature identification is required for accessing each library of at least one of the APIs.

106. (New) The method of claim 104, wherein the APIs of the application platform access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

107. (New) The method of claim 104, wherein the digital signature is generated using a private signature key under the signature scheme, and a public signature key is used to authenticate the digital signature.

108. (New) The method of claim 107, wherein:

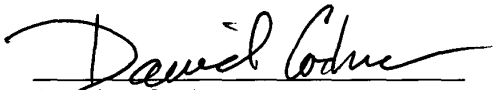
the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the digital signature is authenticated by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

109. (New) The method of claim 104, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

Respectfully submitted,



David B. Cochran
Reg. No. 39,142
Jones, Day
North Point
901 Lakeside Avenue
Cleveland, OH 44114-1190

Code Signing System And Method

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority from and is related to the following prior applications: "Code Signing System And Method," United States Provisional Application No. 60/234,152, filed September 21, 2000; "Code Signing System And Method," United States Provisional Application No. 60/235,354, filed September 26, 2000; and "Code Signing System And Method," United States Provisional Application No. 60/270,663, filed February 20, 2001.

BACKGROUND

1. FIELD OF THE INVENTION

This invention relates generally to the field of security protocols for software applications. More particularly, the invention provides a code signing system and method that is particularly well suited for Java™ applications for mobile communication devices, such as Personal Digital Assistants, cellular telephones, and wireless two-way communication devices (collectively referred to hereinafter as "mobile devices" or simply "devices").

2. DESCRIPTION OF THE RELATED ART

Security protocols involving software code signing schemes are known. Typically, such security protocols are used to ensure the reliability of software applications that are downloaded from the Internet. In a typical software code signing scheme, a digital signature is attached to a software application that identifies the software developer. Once the software is downloaded by a user, the user typically must use his or her judgment to determine whether or not the software

application is reliable, based solely on his or her knowledge of the software developer's reputation. This type of code signing scheme does not ensure that a software application written by a third party for a mobile device will properly interact with the device's native applications and other resources. Because typical code signing protocols are not secure and rely solely on the judgment of the user, there is a serious risk that destructive, "Trojan horse" type software applications may be downloaded and installed onto a mobile device.

There also remains a need for network operators to have a system and method to maintain control over which software applications are activated on mobile devices.

There remains a further need in 2.5G and 3G networks where corporate clients or network operators would like to control the types of software on the devices issued to its employees.

SUMMARY

A code signing system and method is provided. The code signing system operates in conjunction with a software application having a digital signature and includes an application platform, an application programming interface (API), and a virtual machine. The API is configured to link the software application with the application platform. The virtual machine verifies the authenticity of the digital signature in order to control access to the API by the software application.

A code signing system for operation in conjunction with a software application having a digital signature, according to another embodiment of the invention comprises an application platform, a plurality of APIs, each configured to link the software application with a resource on

the application platform, and a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application, wherein the virtual machine verifies the authenticity of the digital signature in order to control access to the plurality of APIs by the software application.

According to a further embodiment of the invention, a method of controlling access to sensitive application programming interfaces on a mobile device comprises the steps of loading a software application on the mobile device that requires access to a sensitive API, determining whether or not the software application includes a digital signature associated with the sensitive API, and if the software application does not include a digital signature associated with the sensitive API, then denying the software application access to the sensitive API.

In another embodiment of the invention, a method of controlling access to an application programming interface (API) on a mobile device by a software application created by a software developer comprises the steps of receiving the software application from the software developer, reviewing the software application to determine if it may access the API, if the software application may access the API, then appending a digital signature to the software application, verifying the authenticity of a digital signature appended to a software application, and providing access to the API to software applications for which the appended digital signature is authentic.

A method of restricting access to a sensitive API on a mobile device, according to a further embodiment of the invention, comprises the steps of registering one or more software developers that are trusted to design software applications which access the sensitive API, receiving a hash of a software application, determining if the software application was designed by one of the registered software developers, and if the software application was designed by one

of the registered software developers, then generating a digital signature using the hash of the software application, wherein the digital signature may be appended to the software application, and the mobile device verifies the authenticity of the digital signature in order to control access to the sensitive API by the software application.

In a still further embodiment, a method of restricting access to application programming interfaces on a mobile device comprises the steps of loading a software application on the mobile device that requires access to one or more API, determining whether or not the software application includes a digital signature associated with the mobile device, and if the software application does not include a digital signature associated with the mobile device, then denying the software application access to the one or more APIs.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram illustrating a code signing protocol according to one embodiment of the invention;

Fig. 2 is a flow diagram of the code signing protocol described above with reference to Fig. 1;

Fig. 3 is a block diagram of a code signing system on a mobile device;

Fig. 3A is a block diagram of a code signing system on a plurality of mobile devices;

Fig. 4 is a flow diagram illustrating the operation of the code signing system described above with reference to Fig. 3 and Fig. 3A;

Fig. 5 is a flow diagram illustrating the management of the code signing authorities described with reference to Fig. 3A; and

Fig. 6 is a block diagram of a mobile communication device in which a code signing system and method may be implemented.

DETAILED DESCRIPTION

Referring now to the drawing figures, Fig. 1 is a diagram illustrating a code signing protocol according to one embodiment of the invention. An application developer 12 creates a software application 14 (application Y) for a mobile device that requires access to one or more sensitive APIs on the mobile device. The software application Y 14 may, for example, be a Java application that operates on a Java virtual machine installed on the mobile device. An API enables the software application Y to interface with an application platform that may include, for example, resources such as the device hardware, operating system and core software and data models. In order to make function calls to or otherwise interact with such device resources, a software application Y must access one or more APIs. APIs can thereby effectively "bridge" a software application and associated device resources. In this description and the appended claims, references to API access should be interpreted to include access of an API in such a way as to allow a software application Y to interact with one or more corresponding device resources. Providing access to any API therefore allows a software application Y to interact with associated device resources, whereas denying access to an API prevents the software application Y from interacting with the associated resources. For example, a database API may communicate with a device file or data storage system, and access to the database API would provide for interaction between a software application Y and the file or data storage system. A user interface (UI) API would communicate with controllers and/or control software for such device components as a

screen, a keyboard, and any other device components that provide output to a user or accept input from a user. In a mobile device, a radio API may also be provided as an interface to wireless communication resources such as a transmitter and receiver. Similarly, a cryptographic API may be provided to interact with a crypto module which implements crypto algorithms on a device. These are merely illustrative examples of APIs that may be provided on a device. A device may include any of these example APIs, or different APIs instead of or in addition to those described above.

Preferably, any API may be classified as sensitive by a mobile device manufacturer, or possibly by an API author, a wireless network operator, a device owner or operator, or some other entity that may be affected by a virus or malicious code in a device software application. For instance, a mobile device manufacturer may classify as sensitive those APIs that interface with cryptographic routines, wireless communication functions, or proprietary data models such as address book or calendar entries. To protect against unauthorized access to these sensitive APIs, the application developer 12 is required to obtain one or more digital signatures from the mobile device manufacturer or other entity that classified any APIs as sensitive, or from a code signing authority 16 acting on behalf of the manufacturer or other entity with an interest in protecting access to sensitive device APIs, and append the signature(s) to the software application Y 14.

In one embodiment, a digital signature is obtained for each sensitive API or library that includes a sensitive API to which the software application requires access. In some cases, multiple signatures are desirable. This would allow a service provider, company or network operator to restrict some or all software applications loaded or updated onto a particular set of

mobile devices. In this multiple-signature scenario, all APIs are restricted and locked until a "global" signature is verified for a software application. For example, a company may wish to prevent its employees from executing any software applications onto their devices without first obtaining permission from a corporate information technology (IT) or computer services department. All such corporate mobile devices may then be configured to require verification of at least a global signature before a software application can be executed. Access to sensitive device APIs and libraries, if any, could then be further restricted, dependent upon verification of respective corresponding digital signatures.

The binary executable representation of software application Y 14 may be independent of the particular type of mobile device or model of a mobile device. Software application Y 14 may for example be in a write-once-run-anywhere binary format such as is the case with Java software applications. However, it may be desirable to have a digital signature for each mobile device type or model, or alternatively for each mobile device platform or manufacturer. Therefore, software application Y 14 may be submitted to several code signing authorities if software application Y 14 targets several mobile devices.

Software application Y 14 is sent from the application developer 12 to the code signing authority 16. In the embodiment shown in Fig. 1, the code signing authority 16 reviews the software application Y 14, although as described in further detail below, it is contemplated that the code signing authority 16 may also or instead consider the identity of the application developer 12 to determine whether or not the software application Y 14 should be signed. The code signing authority 16 is preferably one or more representatives from the mobile device

manufacturer, the authors of any sensitive APIs, or possibly others that have knowledge of the operation of the sensitive APIs to which the software application needs access.

If the code signing authority 16 determines that software application Y 14 may access the sensitive API and therefore should be signed, then a signature (not shown) for the software application Y 14 is generated by the code signing authority 16 and appended to the software application Y 14. The signed software application Y 22, comprising the software application Y 14 and the digital signature, is then returned to the application developer 12. The digital signature is preferably a tag that is generated using a private signature key 18 maintained solely by the code signing authority 16. For example, according to one signature scheme, a hash of the software application Y 14 may be generated, using a hashing algorithm such as the Secure Hash Algorithm SHA1, and then used with the private signature key 18 to create the digital signature. In some signature schemes, the private signature key is used to encrypt a hash of information to be signed, such as software application Y 14, whereas in other schemes, the private key may be used in other ways to generate a signature from the information to be signed or a transformed version of the information.

The signed software application Y 22 may then be sent to a mobile device 28 or downloaded by the mobile device 28 over a wireless network 24. It should be understood, however, that a code signing protocol according to the present invention is not limited to software applications that are downloaded over a wireless network. For instance, in alternative embodiments, the signed software application Y 22 may be downloaded to a personal computer via a computer network and loaded to the mobile device through a serial link, or may be acquired from the application developer 12 in any other manner and loaded onto the mobile device. Once

the signed software application Y 22 is loaded on the mobile device 28, each digital signature is preferably verified with a public signature key 20 before the software application Y 14 is granted access to a sensitive API library. Although the signed software application Y 22 is loaded onto a device, it should be appreciated that the software application that may eventually be executed on the device is the software application Y 14. As described above, the signed software application Y 22 includes the software application Y 14 and one or more appended digital signatures (not shown). When the signatures are verified, the software application Y 14 can be executed on the device and access any APIs for which corresponding signatures have been verified.

The public signature key 20 corresponds to the private signature key 18 maintained by the code signing authority 16, and is preferably installed on the mobile device along with the sensitive API. However, the public key 10 may instead be obtained from a public key repository (not shown), using the device 28 or possibly a personal computer system, and installed on the device 28 as needed. According to one embodiment of a signature scheme, the mobile device 28 calculates a hash of the software application Y 14 in the signed software application Y 22, using the same hashing algorithm as the code signing authority 16, and uses the digital signature and the public signature key 20 to recover the hash calculated by the signing authority 16. The resultant locally calculated hash and the hash recovered from the digital signature are then compared, and if the hashes are the same, the signature is verified. The software application Y 14 can then be executed on the device 28 and access any sensitive APIs for which the corresponding signature(s) have been verified. As described above, the invention is in no way limited to this particular illustrative example signature scheme. Other signature schemes,

including further public key signature schemes, may also be used in conjunction with the code signing methods and systems described herein.

Fig. 2 is a flow diagram 30 of the code signing protocol described above with reference to Fig. 1. The protocol begins at step 32. At step 34, a software developer writes the software application Y for a mobile device that requires access to a sensitive API or library that exposes a sensitive API (API library A). As discussed above, some or all APIs on a mobile device may be classified as sensitive, thus requiring verification of a digital signature for access by any software application such as software application Y. In step 36, application Y is tested by the software developer, preferably using a device simulator in which the digital signature verification function has been disabled. In this manner, the software developer may debug the software application Y before the digital signature is acquired from the code signing authority. Once the software application Y has been written and debugged, it is forwarded to the code signing authority in step 38.

In steps 40 and 42, the code signing authority reviews the software application Y to determine whether or not it should be given access to the sensitive API, and either accepts or rejects the software application. The code signing authority may apply a number of criteria to determine whether or not to grant the software application access to the sensitive API including, for example, the size of the software application, the device resources accessed by the API, the perceived utility of the software application, the interaction with other software applications, the inclusion of a virus or other destructive code, and whether or not the developer has a contractual obligation or other business arrangement with the mobile device manufacturer. Further details of managing code signing authorities and developers are described below in reference to Fig. 5.

If the code signing authority accepts the software application Y, then a digital signature, and preferably a signature identification, are appended to the software application Y in step 46. As described above, the digital signature may be generated by using a hash of the software application Y and a private signature key 18. The signature identification is described below with reference to Figs. 3 and 4. Once the digital signature and signature identification are appended to the software application Y to generate a signed software application, the signed software application Y is returned to the software developer in step 48. The software developer may then license the signed software application Y to be loaded onto a mobile device (step 50). If the code signing authority rejects the software application Y, however, then a rejection notification is preferably sent to the software developer (step 44), and the software application Y will be unable to access any API(s) associated with the signature.

In an alternative embodiment, the software developer may provide the code signing authority with only a hash of the software application Y, or provide the software application Y in some type of abridged format. If the software application Y is a Java application, then the device independent binary *.class files may be used in the hashing operation, although device dependent files such as *.cod files used by the assignee of the present application may instead be used in hashing or other digital signature operations when software applications are intended for operation on particular devices or device types. By providing only a hash or abridged version of the software application Y, the software developer may have the software application Y signed without revealing proprietary code to the code signing authority. The hash of the software application Y, along with the private signature key 18, may then be used by the code signing authority to generate the digital signature. If an otherwise abridged version of the software

application Y is sent to the code signing authority, then the abridged version may similarly be used to generate the digital signature, provided that the abridging scheme or algorithm, like a hashing algorithm, generates different outputs for different inputs. This ensures that every software application will have a different abridged version and thus a different signature that can only be verified when appended to the particular corresponding software application from which the abridged version was generated. Because this embodiment does not enable the code signing authority to thoroughly review the software application for viruses or other destructive code, however, a registration process between the software developer and the code signing authority may also be required. For instance, the code signing authority may agree in advance to provide a trusted software developer access to a limited set of sensitive APIs.

In still another alternative embodiment, a software application Y may be submitted to more than one signing authority. Each signing authority may for example be responsible for signing software applications for particular sensitive APIs or APIs on a particular model of mobile device or set of mobile devices that supports the sensitive APIs required by a software application. A manufacturer, mobile communication network operator, service provider, or corporate client for example may thereby have signing authority over the use of sensitive APIs for their particular mobile device model(s), or the mobile devices operating on a particular network, subscribing to one or more particular services, or distributed to corporate employees. A signed software application may then include a software application and at least one appended digital signature appended from each of the signing authorities. Even though these signing authorities in this example would be generating a signature for the same software application,

different signing and signature verification schemes may be associated with the different signing authorities.

Fig. 3 is a block diagram of a code signing system 60 on a mobile device 62. The system 60 includes a virtual machine 64, a plurality of software applications 66-70, a plurality of API libraries 72-78, and an application platform 80. The application platform 80 preferably includes all of the resources on the mobile device 62 that may be accessed by the software applications 66-70. For instance, the application platform may include device hardware 82, the mobile device's operating system 84, or core software and data models 86. Each API library 72-78 preferably includes a plurality of APIs that interface with a resource available in the application platform. For instance, one API library might include all of the APIs that interface with a calendar program and calendar entry data models. Another API library might include all of the APIs that interface with the transmission circuitry and functions of the mobile device 62. Yet another API library might include all of the APIs capable of interfacing with lower-level services performed by the mobile device's operating system 84. In addition, the plurality of API libraries 72-78 may include both libraries that expose a sensitive API 74 and 78, such as an interface to a cryptographic function, and libraries 72 and 76, that may be accessed without exposing sensitive APIs. Similarly, the plurality of software applications 66-70 may include both signed software applications 66 and 70 that require access to one or more sensitive APIs, and unsigned software applications such as 68. The virtual machine 64 is preferably an object oriented run-time environment such as Sun Micro System's J2ME™ (Java 2 Platform, Micro Edition), which manages the execution of all of the software applications 66-70 operating on the mobile device 62, and links the software applications 66-70 to the various API libraries 72-78.

Software application Y 70 is an example of a signed software application. Each signed software application preferably includes an actual software application such as software application Y comprising for example software code that can be executed on the application platform 80, one or more signature identifications 94 and one or more corresponding digital signatures 96. Preferably each digital signature 96 and associated signature identification 94 in a signed software application 66 or 70 corresponds to a sensitive API library 74 or 78 to which the software application X or software application Y requires access. The sensitive API library 74 or 78 may include one or more sensitive APIs. In an alternative embodiment, the signed software applications 66 and 70 may include a digital signature 96 for each sensitive API within an API library 74 or 78. The signature identifications 94 may be unique integers or some other means of relating a digital signature 96 to a specific API library 74 or 78, API, application platform 80, or model of mobile device 62.

API library A 78 is an example of an API library that exposes a sensitive API. Each API library 74 and 78 including a sensitive API should preferably include a description string 88, a public signature key 20, and a signature identifier 92. The signature identifier 92 preferably corresponds to a signature identification 94 in a signed software application 66 or 70, and enables the virtual machine 64 to quickly match a digital signature 96 with an API library 74 or 78. The public signature key 20 corresponds to the private signature key 18 maintained by the code signing authority, and is used to verify the authenticity of a digital signature 96. The description string 88 may for example be a textual message that is displayed on the mobile device when a signed software application 66 or 70 is loaded, or alternatively when a software application X or Y attempts to access a sensitive API.

Operationally, when a signed software application 68-70, respectively including a software application X, Z, or Y, that requires access to a sensitive API library 74 or 78 is loaded onto a mobile device, the virtual machine 64 searches the signed for an appended digital signature 96 associated with the API library 74 or 78. Preferably, the appropriate digital signature 96 is located by the virtual machine 64 by matching the signature identifier 92 in the API library 74 or 78 with a signature identification 94 on the signed software application. If the signed software application includes the appropriate digital signature 96, then the virtual machine 64 verifies its authenticity using the public signature key 20. Then, once the appropriate digital signature 96 has been located and verified, the description string 88 is preferably displayed on the mobile device before the software application X or Y is executed and accesses the sensitive API. For instance, the description string 88 may display a message stating that "Application Y is attempting to access API Library A," and thereby provide the mobile device user with the final control to grant or deny access to the sensitive API.

Fig. 3A is a block diagram of a code signing system 61 on a plurality of mobile devices 62E, 62F and 62G. The system 61 includes a plurality of mobile devices each of which only three are illustrated, mobile devices 62E, 62F and 62G. Also shown is a signed software application 70, including a software application Y to which two digital signatures 96E and 96F with corresponding signature identifications 94E and 94F have been appended. In the example system 61, each pair composed of a digital signature and identification, 94E/96E and 94F/96F, corresponds to a model of mobile device 62, API library 78, or associated platform 80. If signature identifications 94E and 94F correspond to different models of mobile device 62, then when a signed software application 70 which includes a software application Y that requires

access to a sensitive API library 78 is loaded onto mobile device 62E, the virtual machine 64 searches the signed software application 70 for a digital signature 96E associated with the API library 78 by matching identifier 94E with signature identifier 92. Similarly, when a signed software application 70 including a software application Y that requires access to a sensitive API library 78 is loaded onto a mobile device 62F, the virtual machine 64 in device 62F searches the signed software application 70 for a digital signature 96F associated with the API library 78. However, when a software application Y in a signed software application 70 that requires access to a sensitive API library 78 is loaded onto a mobile device model for which the application developer has not obtained a digital signature, device 62G in the example of Fig. 3A, the virtual machine 64 in the device 64G does not find a digital signature appended to the software application Y and consequently, access to the API library 78 is denied on device 62G. It should be appreciated from the foregoing description that a software application such as software application Y may have multiple device-specific, library-specific, or API-specific signatures or some combination of such signatures appended thereto. Similarly, different signature verification requirements may be configured for the different devices. For example, device 62E may require verification of both a global signature, as well as additional signatures for any sensitive APIs to which a software application requires access in order for the software application to be executed, whereas device 62F may require verification of only a global signature and device 62G may require verification of signatures only for its sensitive APIs. It should also be apparent that a communication system may include devices (not shown) on which a software application Y received as part of a signed software application such as 70 may execute without any signature verification. Although a signed software application has one or

more signatures appended thereto, the software application Y might possibly be executed on some devices without first having any of its signature(s) verified. Signing of a software application preferably does not interfere with its execution on devices in which digital signature verification is not implemented.

Fig. 4 is a flow diagram 100 illustrating the operation of the code signing system described above with reference to Figs. 3 and 3A. In step 102, a software application is loaded onto a mobile device. Once the software application is loaded, the device, preferably using a virtual machine, determines whether or not the software application requires access to any API libraries that expose a sensitive API (step 104). If not, then the software application is linked with all of its required API libraries and executed (step 118). If the software application does require access to a sensitive API, however, then the virtual machine verifies that the software application includes a valid digital signature associated any sensitive APIs to which access is required, in steps 106-116.

In step 106, the virtual machine retrieves the public signature key 20 and signature identifier 92 from the sensitive API library. The signature identifier 92 is then used by the virtual machine in step 108 to determine whether or not the software application has an appended digital signature 96 with a corresponding signature identification 94. If not, then the software application has not been approved for access to the sensitive API by a code signing authority, and the software application is preferably prevented from being executed in step 116. In alternative embodiments, a software application without a proper digital signature 96 may be purged from the mobile device, or may be denied access to the API library exposing the sensitive API but executed to the extent possible without access to the API library. It is also contemplated

that a user may be prompted for an input when signature verification fails, thereby providing for user control of such subsequent operations as purging of the software application from the device.

If a digital signature 96 corresponding to the sensitive API library is appended to the software application and located by the virtual machine, then the virtual machine uses the public key 20 to verify the authenticity of the digital signature 96 in step 110. This step may be performed, for example, by using the signature verification scheme described above or other alternative signature schemes. If the digital signature 96 is not authentic, then the software application is preferably either not executed, purged, or restricted from accessing the sensitive API as described above with reference to step 116. If the digital signature is authentic, however, then the description string 88 is preferably displayed in step 112, warning the mobile device user that the software application requires access to a sensitive API, and possibly prompting the user for authorization to execute or load the software application (step 114). When more than one signature is to be verified for a software application, then the steps 104-110 are preferably repeated for each signature before the user is prompted in step 112. If the mobile device user in step 114 authorizes the software application, then it may be executed and linked to the sensitive API library in step 118.

Fig. 5 is a flow diagram 200 illustrating the management of the code signing authorities described with reference to Fig. 3A. At step 210, an application developer has developed a new software application which is intended to be executable one or more target device models or types. The target devices may include sets of devices from different manufacturers, sets of device models or types from the same manufacturer, or generally any sets of devices having

particular signature and verification requirements. The term "target device" refers to any such set of devices having a common signature requirement. For example, a set of devices requiring verification of a device-specific global signature for execution of all software applications may comprise a target device, and devices that require both a global signature and further signatures for sensitive APIs may be part of more than one target device set. The software application may be written in a device independent manner by using at least one known API, supported on at least one target device with an API library. Preferably, the developed software application is intended to be executable on several target devices, each of which has its own at least one API library.

At step 220, a code signing authority for one target device receives a target-signing request from the developer. The target signing request includes the software application or a hash of the software application, a developer identifier, as well as at least one target device identifier which identifies the target device for which a signature is being requested. At step 230, the signing authority consults a developer database 235 or other records to determine whether or not to trust developer 220. This determination can be made according to several criteria discussed above, such as whether or not the developer has a contractual obligation or has entered into some other type of business arrangement with a device manufacturer, network operator, service provider, or device manufacturer. If the developer is trusted, then the method proceeds at step 240. However, if the developer is not trusted, then the software application is rejected (250) and not signed by the signing authority. Assuming the developer was trusted, at step 240 the signing authority determines if it has the target private key corresponding to the submitted target identifier by consulting a private key store such as a target private key database 245. If the target private key is found, then a digital signature for the software application is generated at step 260

and the digital signature or a signed software application including the digital signature appended to the software application is returned to the developer at step 280. However, if the target private key is not found at step 240, then the software application is rejected at step 270 and no digital signature is generated for the software application.

Advantageously, if target signing authorities follow compatible embodiments of the method outlined in Fig. 5, a network of target signing authorities may be established in order to expediently manage code signing authorities and a developer community code signing process providing signed software applications for multiple targets with low likelihood of destructive code.

Should any destructive or otherwise problematic code be found in a software application or suspected because of behavior exhibited when a software application is executed on a device, then the registration or privileges of the corresponding application developer with any or all signing authorities may also be suspended or revoked, since the digital signature provides an audit trail through which the developer of a problematic software application may be identified. In such an event, devices may be informed of the revocation by being configured to periodically download signature revocation lists, for example. If software applications for which the corresponding digital signatures have been revoked are running on a device, the device may then halt execution of any such software application and possibly purge the software application from its local storage. If preferred, devices may also be configured to re-execute digital signature verifications, for instance periodically or when a new revocation list is downloaded.

Although a digital signature generated by a signing authority is dependent upon authentication of the application developer and confirmation that the application developer has

been properly registered, the digital signature is preferably generated from a hash or otherwise transformed version of the software application and is therefore application-specific. This contrasts with known code signing schemes, in which API access is granted to any software applications arriving from trusted application developers or authors. In the code signing systems and methods described herein, API access is granted on an application-by-application basis and thus can be more strictly controlled or regulated.

Fig. 6 is a block diagram of a mobile communication device in which a code signing system and method may be implemented. The mobile communication device 610 is preferably a two-way communication device having at least voice and data communication capabilities. The device preferably has the capability to communicate with other computer systems on the Internet. Depending on the functionality provided by the device, the device may be referred to as a data messaging device, a two-way pager, a cellular telephone with data messaging capabilities, a wireless Internet appliance or a data communication device (with or without telephony capabilities).

Where the device 610 is enabled for two-way communications, the device will incorporate a communication subsystem 611, including a receiver 612, a transmitter 614, and associated components such as one or more, preferably embedded or internal, antenna elements 616 and 618, local oscillators (LOs) 613, and a processing module such as a digital signal processor (DSP) 620. As will be apparent to those skilled in the field of communications, the particular design of the communication subsystem 611 will be dependent upon the communication network in which the device is intended to operate. For example, a device 610 destined for a North American market may include a communication subsystem 611 designed to

operate within the Mobitex™ mobile communication system or DataTAC™ mobile communication system, whereas a device 610 intended for use in Europe may incorporate a General Packet Radio Service (GPRS) communication subsystem 611.

Network access requirements will also vary depending upon the type of network 919. For example, in the Mobitex and DataTAC networks, mobile devices such as 610 are registered on the network using a unique identification number associated with each device. In GPRS networks however, network access is associated with a subscriber or user of a device 610. A GPRS device therefore requires a subscriber identity module (not shown), commonly referred to as a SIM card, in order to operate on a GPRS network. Without a SIM card, a GPRS device will not be fully functional. Local or non-network communication functions (if any) may be operable, but the device 610 will be unable to carry out any functions involving communications over network 619, other than any legally required operations such as "911" emergency calling.

When required network registration or activation procedures have been completed, a device 610 may send and receive communication signals over the network 619. Signals received by the antenna 616 through a communication network 619 are input to the receiver 612, which may perform such common receiver functions as signal amplification, frequency down conversion, filtering, channel selection and the like, and in the example system shown in Fig. 6, analog to digital conversion. Analog to digital conversion of a received signal allows more complex communication functions such as demodulation and decoding to be performed in the DSP 620. In a similar manner, signals to be transmitted are processed, including modulation and encoding for example, by the DSP 620 and input to the transmitter 614 for digital to analog

conversion, frequency up conversion, filtering, amplification and transmission over the communication network 619 via the antenna 618.

The DSP 620 not only processes communication signals, but also provides for receiver and transmitter control. For example, the gains applied to communication signals in the receiver 612 and transmitter 614 may be adaptively controlled through automatic gain control algorithms implemented in the DSP 620.

The device 610 preferably includes a microprocessor 638 which controls the overall operation of the device. Communication functions, including at least data and voice communications, are performed through the communication subsystem 611. The microprocessor 638 also interacts with further device subsystems or resources such as the display 622, flash memory 624, random access memory (RAM) 626, auxiliary input/output (I/O) subsystems 628, serial port 630, keyboard 632, speaker 634, microphone 636, a short-range communications subsystem 640 and any other device subsystems generally designated as 642. APIs, including sensitive APIs requiring verification of one or more corresponding digital signatures before access is granted, may be provided on the device 610 to interface between software applications and any of the resources shown in Fig. 6.

Some of the subsystems shown in Fig. 6 perform communication-related functions, whereas other subsystems may provide "resident" or on-device functions. Notably, some subsystems, such as keyboard 632 and display 622 for example, may be used for both communication-related functions, such as entering a text message for transmission over a communication network, and device-resident functions such as a calculator or task list.

Operating system software used by the microprocessor 638, and possibly APIs to be accessed by software applications, is preferably stored in a persistent store such as flash memory 624, which may instead be a read only memory (ROM) or similar storage element (not shown). Those skilled in the art will appreciate that the operating system, specific device software applications, or parts thereof, may be temporarily loaded into a volatile store such as RAM 626. It is contemplated that received and transmitted communication signals may also be stored to RAM 626.

The microprocessor 638, in addition to its operating system functions, preferably enables execution of software applications on the device. A predetermined set of applications which control basic device operations, including at least data and voice communication applications for example, will normally be installed on the device 610 during manufacture. A preferred application that may be loaded onto the device may be a personal information manager (PIM) application having the ability to organize and manage data items relating to the device user such as, but not limited to e-mail, calendar events, voice mails, appointments, and task items. Naturally, one or more memory stores would be available on the device to facilitate storage of PIM data items on the device. Such PIM application would preferably have the ability to send and receive data items, via the wireless network. In a preferred embodiment, the PIM data items are seamlessly integrated, synchronized and updated, via the wireless network, with the device user's corresponding data items stored or associated with a host computer system thereby creating a mirrored host computer on the mobile device with respect to the data items at least. This would be especially advantageous in the case where the host computer system is the mobile device user's office computer system. Further applications, including signed software

applications as described above, may also be loaded onto the device 610 through the network 619, an auxiliary I/O subsystem 628, serial port 630, short-range communications subsystem 640 or any other suitable subsystem 642. The device microprocessor 638 may then verify any digital signatures, possibly including both "global" device signatures and API-specific signatures, appended to such a software application before the software application can be executed by the microprocessor 638 and/or access any associated sensitive APIs. Such flexibility in application installation increases the functionality of the device and may provide enhanced on-device functions, communication-related functions, or both. For example, secure communication applications may enable electronic commerce functions and other such financial transactions to be performed using the device 610, through a crypto API and a crypto module which implements crypto algorithms on the device (not shown).

In a data communication mode, a received signal such as a text message or web page download will be processed by the communication subsystem 611 and input to the microprocessor 638, which will preferably further process the received signal for output to the display 622, or alternatively to an auxiliary I/O device 628. A user of device 610 may also compose data items such as email messages for example, using the keyboard 632, which is preferably a complete alphanumeric keyboard or telephone-type keypad, in conjunction with the display 622 and possibly an auxiliary I/O device 628. Such composed items may then be transmitted over a communication network through the communication subsystem 611.

For voice communications, overall operation of the device 610 is substantially similar, except that received signals would preferably be output to a speaker 634 and signals for transmission would be generated by a microphone 636. Alternative voice or audio I/O

subsystems such as a voice message recording subsystem may also be implemented on the device 610. Although voice or audio signal output is preferably accomplished primarily through the speaker 634, the display 622 may also be used to provide an indication of the identity of a calling party, the duration of a voice call, or other voice call related information for example.

The serial port 630 in Fig. 6 would normally be implemented in a personal digital assistant (PDA)-type communication device for which synchronization with a user's desktop computer (not shown) may be desirable, but is an optional device component. Such a port 630 would enable a user to set preferences through an external device or software application and would extend the capabilities of the device by providing for information or software downloads to the device 610 other than through a wireless communication network. The alternate download path may for example be used to load an encryption key onto the device through a direct and thus reliable and trusted connection to thereby enable secure device communication.

A short-range communications subsystem 640 is a further optional component which may provide for communication between the device 624 and different systems or devices, which need not necessarily be similar devices. For example, the subsystem 640 may include an infrared device and associated circuits and components or a Bluetooth™ communication module to provide for communication with similarly-enabled systems and devices.

The embodiments described herein are examples of structures, systems or methods having elements corresponding to the elements of the invention recited in the claims. This written description may enable those skilled in the art to make and use embodiments having alternative elements that likewise correspond to the elements of the invention recited in the claims. The intended scope of the invention thus includes other structures, systems or methods

that do not differ from the literal language of the claims, and further includes other structures, systems or methods with insubstantial differences from the literal language of the claims.

For example, when a software application is rejected at step 250 in the method shown in Fig. 5, the signing authority may request that the developer sign a contract or enter into a business relationship with a device manufacturer or other entity on whose behalf the signing authority acts. Similarly, if a software application is rejected at step 270, authority to sign the software application may be delegated to a different signing authority. The signing of a software application following delegation of signing of the software application to the different authority can proceed substantially as shown in Fig. 5, wherein the target signing authority that received the original request from the trusted developer at step 220 requests that the software application be signed by the different signing authority on behalf of the trusted developer from the target signing authority. Once a trust relationship has been established between code signing authorities, target private code signing keys could be shared between code signing authorities to improve performance of the method at step 240, or a device may be configured to validate digital signatures from either of the trusted signing authorities.

In addition, although described primarily in the context of software applications, code signing systems and methods according to the present invention may also be applied to other device-related components, including but in no way limited to, commands and associated command arguments, and libraries configured to interface with device resources. Such commands and libraries may be sent to mobile devices by device manufacturers, device owners, network operators, service providers, software application developers and the like. It would be desirable to control the execution of any command that may affect device operation, such as a

command to change a device identification code or wireless communication network address for example, by requiring verification of one or more digital signatures before a command can be executed on a device, in accordance with the code signing systems and methods described and claimed herein.

We claim:

1. A code signing system for operation in conjunction with a software application having a digital signature, comprising:

an application platform;

an application programming interface (API) configured to link the software application with the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application.

2. The code signing system of claim 1, wherein the virtual machine denies the software application access to the API if the digital signature is not authentic.

3. The code signing system of claim 1, wherein the virtual machine purges the software application if the digital signature is not authentic.

4. The code signing system of claim 1, wherein the code signing system is installed on a mobile device.

5. The code signing system of claim 1, wherein the digital signature is generated by a code signing authority.

6. A code signing system for operation in conjunction with a software application having a digital signature, comprising:

an application platform;

a plurality of application programming interfaces (APIs), each configured to link the software application with a resource on the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application,

wherein the virtual machine verifies the authenticity of the digital signature in order to control access to the plurality of APIs by the software application.

7. The code signing system of claim 6, wherein the plurality of APIs are included in an API library.

8. The code signing system of claim 6, wherein one or more of the plurality of APIs is classified as sensitive, and wherein the virtual machine uses the digital signature to control access to the sensitive APIs.

9. The code signing system of claim 8, for operation in conjunction with a plurality of software applications, wherein one or more of the plurality of software applications has a digital signature, and wherein the virtual machine verifies the authenticity of the digital signature of each of the one or more of the plurality of software applications in order to control access to the sensitive APIs by each of the plurality of software applications.

10. The code signing system of claim 6, wherein the resource on the application platform comprises a wireless communication system.

11. The code signing system of claim 6, wherein the resource on the application platform comprises a cryptographic module which implements cryptographic algorithms.

12. The code signing system of claim 6, wherein the resource on the application platform comprises a data store.

13. The code signing system of claim 6, wherein the resource on the application platform comprises a user interface (UI).

14. The code signing system of claim 1, further comprising:

a plurality of API libraries each including a plurality of APIs, wherein the virtual machine controls access to the plurality of API libraries by the software application.

15. The code signing system of claim 14, wherein one or more of the plurality of API libraries is classified as sensitive, and wherein the virtual machine uses the digital signature to control access to the sensitive API libraries by the software application.

16. The code signing system of claim 15, wherein the software application includes a unique digital signature for each sensitive API library.

17. The code signing system of claim 16, wherein:

the software application includes a signature identification for each unique digital signature;

each sensitive API library includes a signature identifier; and

the virtual machine compares the signature identification and the signature identifier to match the unique digital signatures with sensitive API libraries.

18. The code signing system of claim 1, wherein the digital signature is generated using a private signature key, and the virtual machine uses a public signature key to verify the authenticity of the digital signature.

19. The code signing system of claim 18, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application; and

the virtual machine verifies the authenticity of the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and comparing the generated hash with the recovered hash.

20. The code signing system of claim 1, wherein the API further comprises:

a description string that is displayed by the mobile device when the software application attempts to access the API.

21. The code signing system of claim 1, wherein the application platform comprises an operating system.

22. The code signing system of claim 1, wherein the application platform comprises one or more core functions of a mobile device.

23. The code signing system of claim 1, wherein the application platform comprises hardware on a mobile device.

24. The code signing system of claim 23, wherein the hardware comprises a subscriber identity module (SIM) card.

25. The code signing system of claim 1, wherein the software application is a Java application for a mobile device.

26. The code signing system of claim 1, wherein the API interfaces with a cryptographic routine on the application platform.

27. The code signing system of claim 1, wherein the API interfaces with a proprietary data model on the application platform.

28. The code signing system of claim 1, wherein the virtual machine is a Java virtual machine installed on a mobile device.

29. A method of controlling access to sensitive application programming interfaces on a mobile device, comprising the steps of:

loading a software application on the mobile device that requires access to a sensitive application programming interface (API);

determining whether or not the software application includes a digital signature associated with the sensitive API; and

if the software application does not include a digital signature associated with the sensitive API, then denying the software application access to the sensitive API.

30. The method of claim 29, comprising the additional step of:

if the software application does not include a digital signature associated with the sensitive API, then purging the software application from the mobile device.

31. The method of claim 29, wherein the digital signature is generated by a code signing authority.

32. The method of claim 29, comprising the additional steps of:

if the software application includes a digital signature associated with the sensitive API, then verifying the authenticity of the digital signature; and

if the digital signature is not authentic, then denying the software application access to the sensitive API.

33. The method of claim 32, comprising the additional step of:

if the digital signature is not authentic, then purging the software application from the mobile device.

34. The method of claim 32, wherein the digital signature is generated by applying a private signature key to a hash of the software application, and wherein the step of verifying the authenticity of the digital signature is performed by a method comprising the steps of:

storing a public signature key that corresponds to the private signature key on the mobile device;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

35. The method of claim 34, wherein the digital signature is generated by calculating a hash of the software application and applying the private signature key.

36. The method of claim 29, comprising the additional step of:

displaying a description string that notifies a user of the mobile device that the software application requires access to the sensitive API.

37. The method of claim 36, comprising the additional step of:

receiving a command from the user granting or denying the software application access to the sensitive API.

38. A method of controlling access to an application programming interface (API) on a mobile device by a software application created by a software developer, comprising the steps of:

receiving the software application from the software developer;

reviewing the software application to determine if it may access the API;

if the software application may access the API, then appending a digital signature to the software application;

verifying the authenticity of a digital signature appended to a software application; and

providing access to the API to software applications for which the appended digital signature is authentic.

39. The method of claim 38, wherein the step of reviewing the software application is performed by a code signing authority.

40. The method of claim 38, wherein the step of appending the digital signature to the software application is performed by a method comprising the steps of:

calculating a hash of the software application; and

applying a signature key to the hash of the software application to generate the digital signature.

41. The method of claim 40, wherein the hash of the software application is calculated using the Secure Hash Algorithm (SHA1).

42. The method of claim 40, wherein the step of verifying the authenticity of a digital signature comprises the steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash; and

determining if the digital signature is authentic by comparing the calculated hash with the recovered hash.

43. The method of claim 42, comprising the further step of, if the digital signature is not authentic, then denying the software application access to the API.

44. The method of claim 42, wherein the signature key is a private signature key and the corresponding signature key is a public signature key.

45. A method of controlling access to a sensitive application programming interface (API) on a mobile device, comprising the steps of:

registering one or more software developers that are trusted to design software applications which access the sensitive API;

receiving a hash of a software application;

determining if the software application was designed by one of the registered software developers; and

if the software application was designed by one of the registered software developers, then generating a digital signature using the hash of the software application,

wherein

the digital signature may be appended to the software application; and

the mobile device verifies the authenticity of the digital signature in order to control access to the sensitive API by the software application.

46. The method of claim 45, wherein the step of generating the digital signature is performed by a code signing authority.

47. The method of claim 45, wherein the step of generating the digital signature is performed by applying a signature key to the hash of the software application.

48. The method of claim 47, wherein the mobile device verifies the authenticity of the digital signature by performing the additional steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash;

determining if the digital signature is authentic by comparing the calculated hash with the recovered hash; and

if the digital signature is not authentic, then denying the software application access to the sensitive API.

49. A method of restricting access to application programming interfaces on a mobile device, comprising the steps of:

loading a software application on the mobile device that requires access to one or more application programming interface (API);

determining whether or not the software application includes an authentic digital signature associated with the mobile device; and

if the software application does not include an authentic digital signature associated with the mobile device, then denying the software application access to the one or more APIs.

50. The method of claim 49, comprising the additional step of:

if the software application does not include an authentic digital signature associated with the mobile device, then purging the software application from the mobile device.

51. The method of claim 49, wherein:

the software application includes a plurality of digital signatures; and

the plurality of digital signatures includes digital signatures respectively associated with different types of mobile devices.

52. The method of claim 51, wherein each of the plurality of digital signatures is generated by a respective corresponding code signing authority.

53. The method of claim 49, wherein the step of determining whether or not the software application includes an authentic digital signature associated with the mobile device comprises the additional steps of:

determining if the software application includes a digital signature associated with the mobile device; and

if so, then verifying the authenticity of the digital signature.

54. The method of claim 53, wherein the one or more APIs includes one or more APIs classified as sensitive, and the method further comprises the steps of, for each sensitive API:

determining whether or not the software application includes an authentic digital signature associated with the sensitive API; and

if the software application does not include an authentic digital signature associated with the sensitive API, then denying the software application access to the sensitive API.

55. The method of claim 52, wherein each of the plurality of digital signatures is generated by its corresponding code signing authority by applying a respective private signature key associated with the code signing authority to a hash of the software application.

56. The method of claim 55, wherein the step of determining whether or not the software application includes an authentic digital signature associated with the mobile device comprises the steps of:

determining if the software application includes a digital signature associated with the mobile device; and

if so, then verifying the authenticity of the digital signature,

wherein the step of verifying the authenticity of the digital signature is performed by a method comprising the steps of:

storing a public signature key on a mobile device that corresponds to the private signature key associated with the code signing authority which generates the signature associated with the mobile device;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
28 March 2002 (28.03.2002)

PCT

(10) International Publication Number
WO 02/25409 A2

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/CA01/01344

(22) International Filing Date:
20 September 2001 (20.09.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/234,152	21 September 2000 (21.09.2000)	US
60/235,354	26 September 2000 (26.09.2000)	US
60/270,663	20 February 2001 (20.02.2001)	US

(71) Applicant *(for all designated States except US)*: **RESEARCH IN MOTION LIMITED** [CA/CA]; 295 Phillip Street, Waterloo, Ontario N2L 3W8 (CA).

(72) Inventors; and

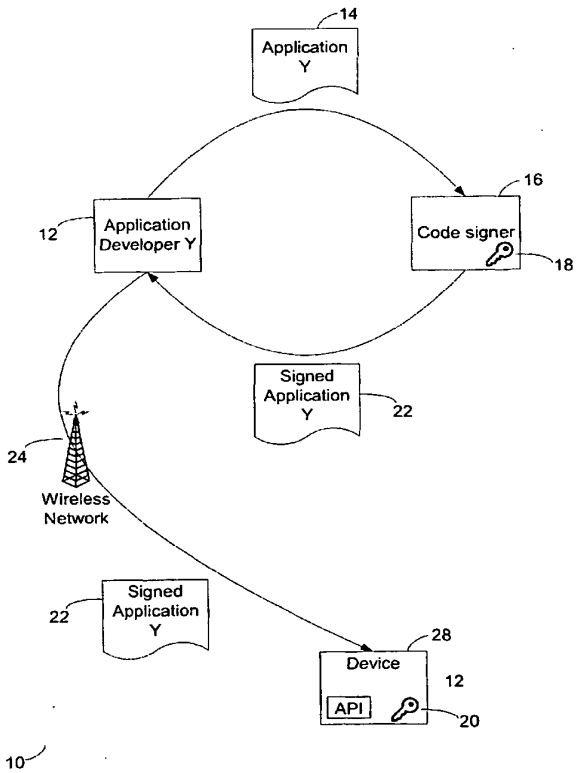
(75) Inventors/Applicants *(for US only)*: **YACH, David, P.** [CA/CA]; 254 Castlefield Avenue, Waterloo, Ontario N2K 2N1 (CA). **BROWN, Michael, S.** [CA/CA]; 7 Danube Street, Heidelberg, Ontario N0B 1Y0 (CA). **LITTLE, Herbert, A.** [CA/CA]; 504 Old Oak Place, Waterloo, Ontario N2T 2V8 (CA).

(74) Agent: **PATHIYAL, Krishna, K.**; Research In Motion Limited, 295 Phillip Street, Waterloo, Ontario N2L 3W8 (CA).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK,

[Continued on next page]

(54) Title: CODE SIGNING SYSTEM AND METHOD



(57) Abstract: A code signing system and method is provided. The code signing system operates in conjunction with a signed software application having a digital signature and includes an application platform, an application programming interface (API), and a virtual machine. The API is configured to link the software application with the application platform. The virtual machine verifies the authenticity of the digital signature in order to control access to the API by the software application.

WO 02/25409 A2

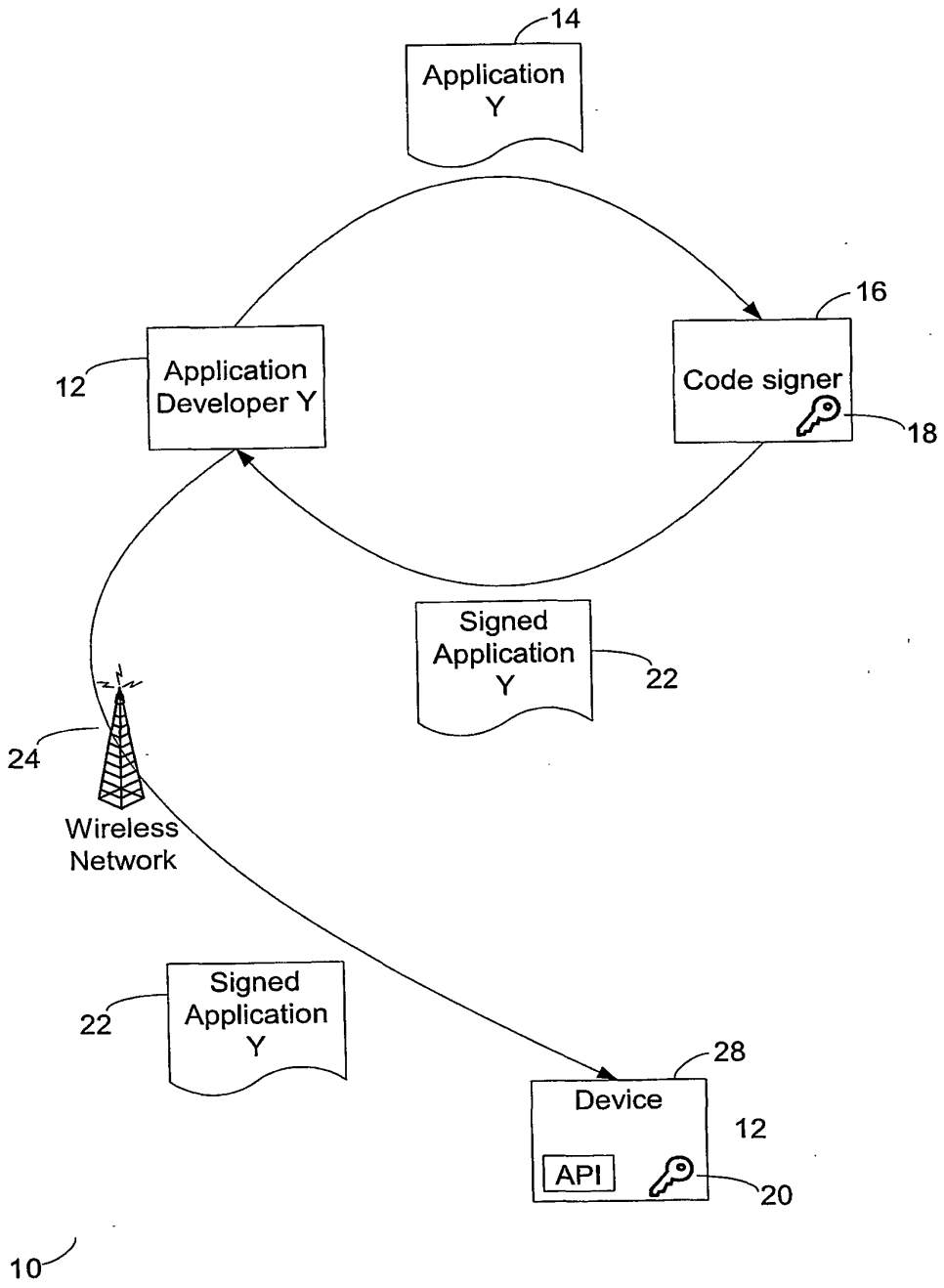
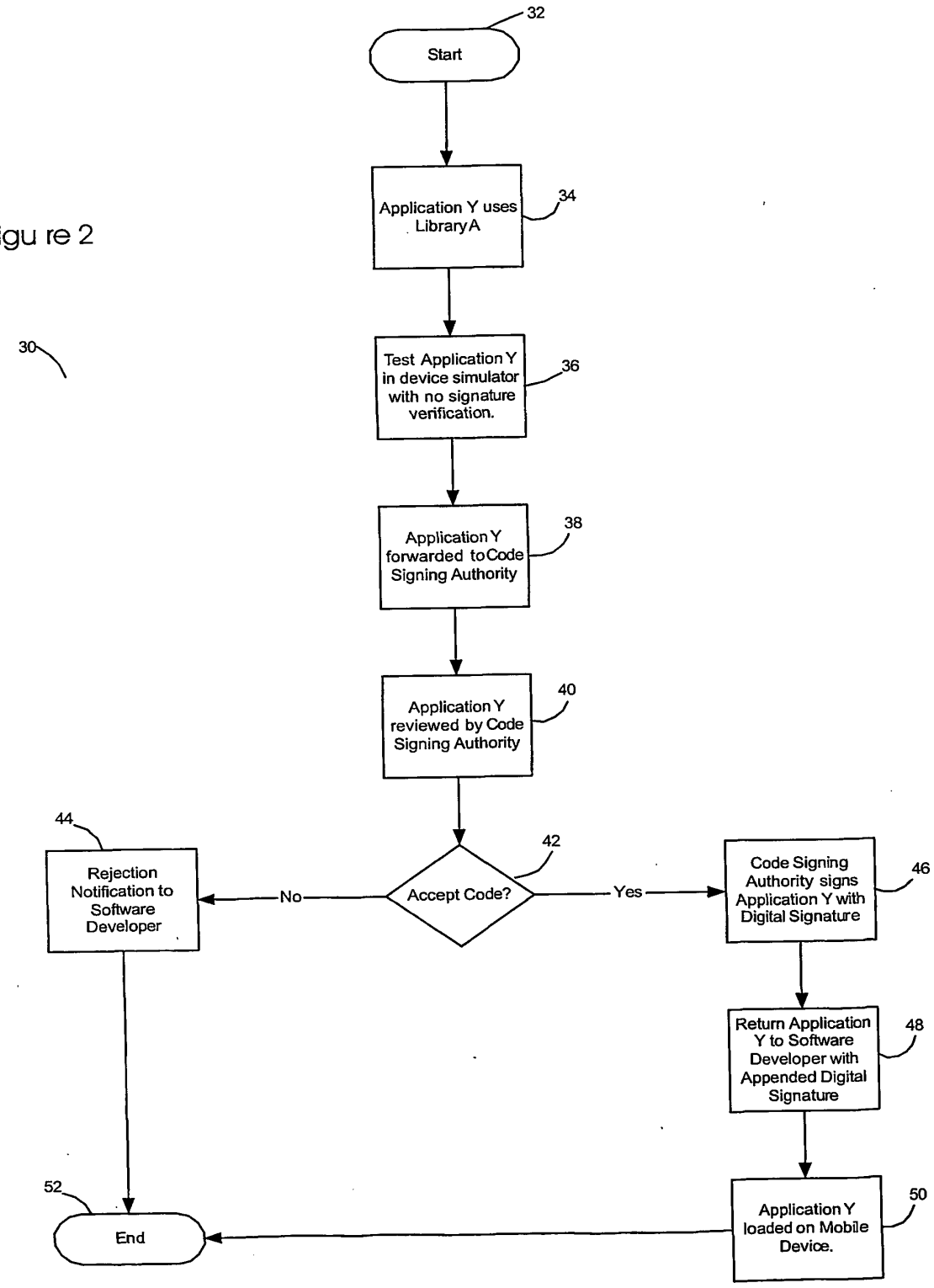
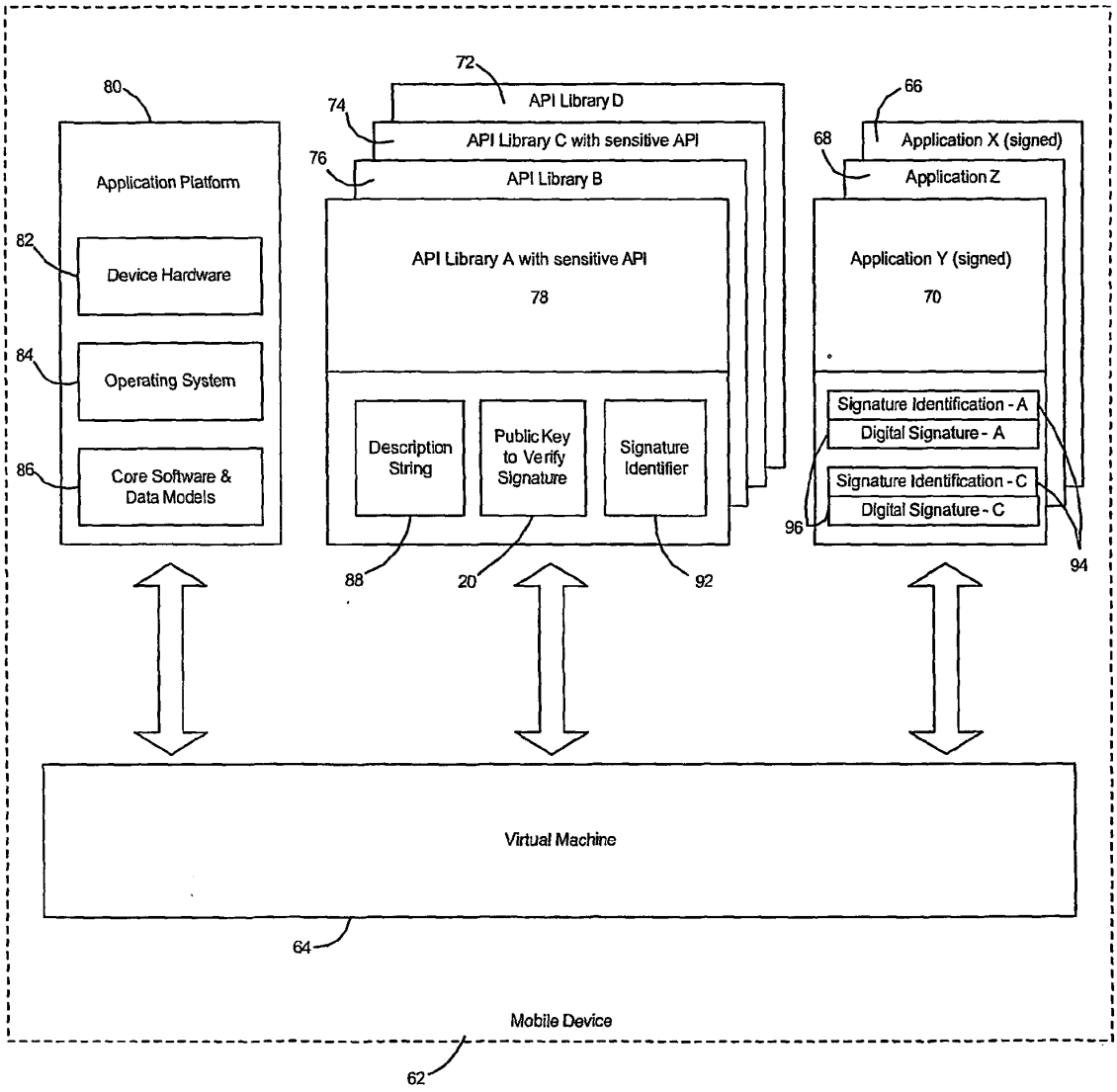

14
Application Y
16
12
Application Developer Y
Code signer
18
Signed Application Y
22
24
Wireless Network
22
Signed Application Y
28
Device
12
API
20
10

Figure 1

Figu re 2


30
32
Start
Application Y uses Library A
34
Test Application Y in device simulator with no signature verification.
36
Application Y forwarded to Code Signing Authority
38
Application Y reviewed by Code Signing Authority
40
Accept Code?
42
No
Yes
44
Rejection Notification to Software Developer
Code Signing Authority signs Application Y with Digital Signature
46
Return Application Y to Software Developer with Appended Digital Signature
48
Application Y loaded on Mobile Device.
50
52
End

3/7


80
Application Platform
82
Device Hardware
84
Operating System
86
Core Software & Data Models
72
API Library D
74
API Library C with sensitive API
76
API Library B
API Library A with sensitive API
78
Description String
Public Key to Verify Signature
Signature Identifier
88
20
92
66
Application X (signed)
68
Application Z
Application Y (signed)
70
Signature Identification - A
Digital Signature - A
Signature Identification - C
Digital Signature - C
96
94
Virtual Machine
64
Mobile Device
62
60

Figu re 3

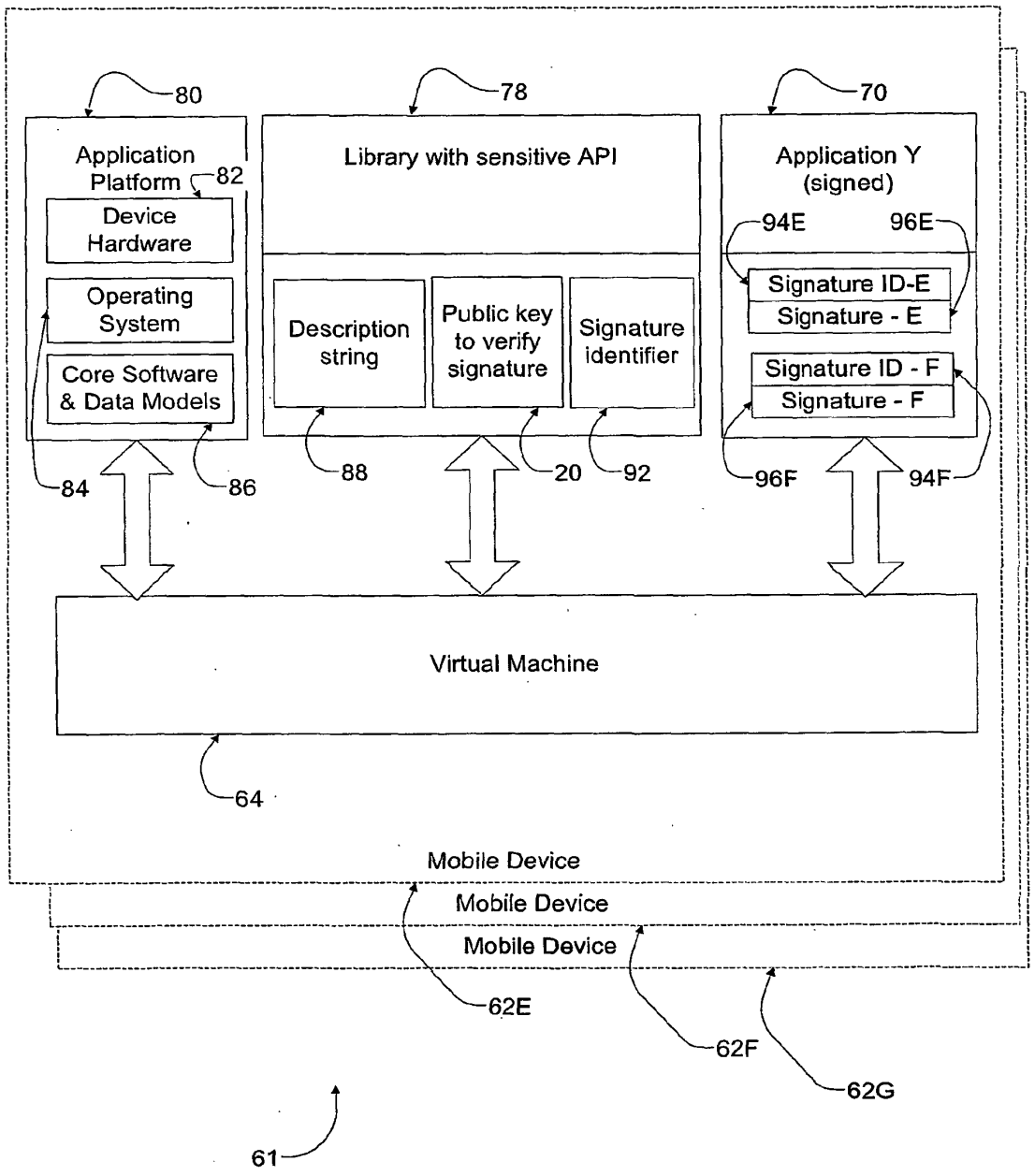
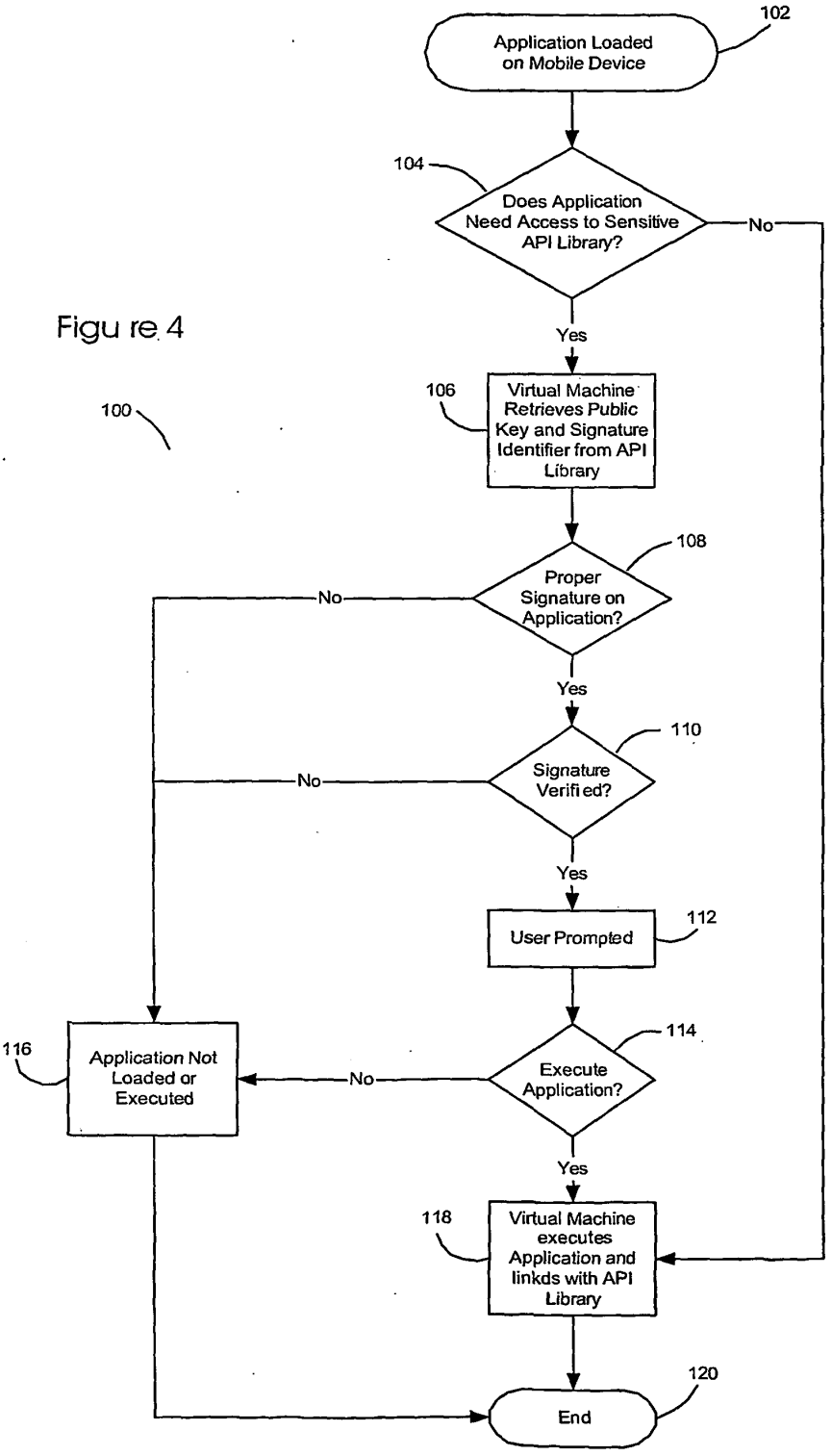

80
78
70
Application Platform
82
Device Hardware
Operating System
Core Software & Data Models
Library with sensitive API
Description string
Public key to verify signature
Signature identifier
Application Y (signed)
94E
96E
Signature ID-E
Signature - E
Signature ID - F
Signature - F
84
86
88
20
92
96F
94F
Virtual Machine
64
Mobile Device
Mobile Device
Mobile Device
62E
62F
62G
61

Figure 3A

Figu re 4


100
102
Application Loaded on Mobile Device
104
Does Application Need Access to Sensitive API Library?
No
Yes
106
Virtual Machine Retrieves Public Key and Signature Identifier from API Library
108
Proper Signature on Application?
No
Yes
110
Signature Verified?
No
Yes
112
User Prompted
114
Execute Application?
No
Yes
116
Application Not Loaded or Executed
118
Virtual Machine executes Application and linkds with API Library
120
End

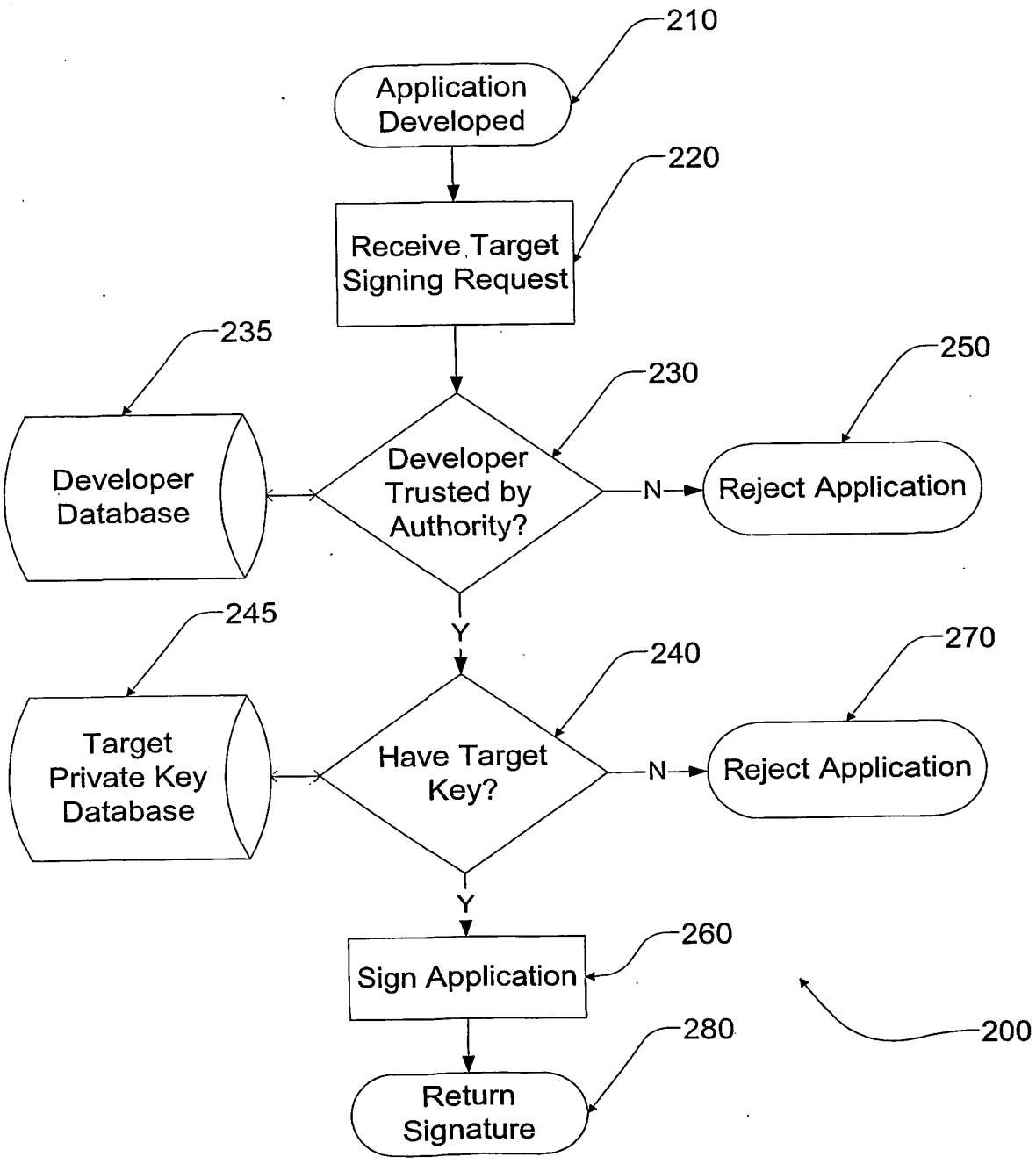

210
Application Developed
220
Receive Target Signing Request
235
Developer Database
230
Developer Trusted by Authority?
N
250
Reject Application
Y
245
Target Private Key Database
240
Have Target Key?
N
270
Reject Application
Y
260
Sign Application
280
Return Signature
200

Figure 5

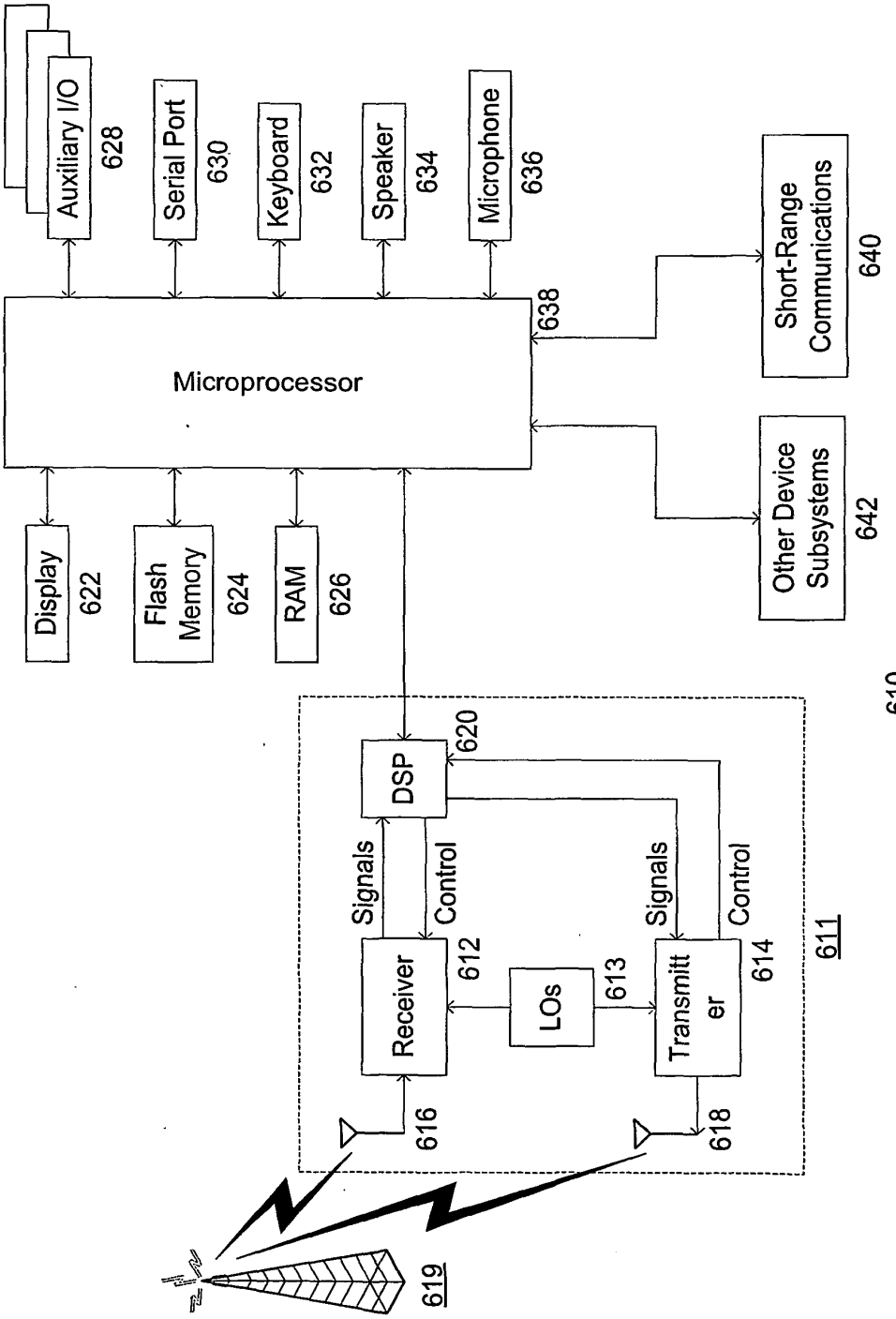

Auxiliary I/O
628
Serial Port
630
Keyboard
632
Speaker
634
Microphone
636
638
Microprocessor
Short-Range Communications
640
Other Device Subsystems
642
Display
622
Flash Memory
624
RAM
626
610
DSP
620
Signals
Control
Signals
Control
Receiver
612
LOs
613
Transmitter
614
611
616
618
619

Figure 6

Express Mail No. EV 243791125 US
March 20, 2003

PTO/SB/01 (03-01)
Approved for use through 10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

DECLARATION FOR UTILITY OR DESIGN PATENT APPLICATION (37 CFR 1.63)

[X] Declaration Submitted with Initial Filing OR [ ] Declaration Submitted after Initial Filing (surcharge (37 CFR 1.16 (e)) required)

Attorney Docket Number	555255012423
First Named Inventor	David P. YACH
COMPLETE IF KNOWN	
Application Number	/
Filing Date	March 20, 2003
Group Art Unit	
Examiner Name	

As a below named inventor, I hereby declare that:

My residence, mailing address, and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

SOFTWARE CODE SIGNING SYSTEM AND METHOD

(Title of the Invention)

the specification of which

[X] is attached hereto

OR

[ ] was filed on (MM/DD/YYYY) [ ] as United States Application Number or PCT International

Application Number [ ] and was amended on (MM/DD/YYYY) [ ] (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment specifically referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56, including for continuation-in-part applications, material information which became available between the filing date of the prior application and the national or PCT international filing date of the continuation-in-part application.

I hereby claim foreign priority benefits under 35 U.S.C. 119(a)-(d) or (f), or 365(b) of any foreign application(s) for patent, inventor's or plant breeder's rights certificate(s), or 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below, by checking the box, any foreign application for patent, inventor's or plant breeder's rights certificate(s), or any PCT international application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application Number(s)	Country	Foreign Filing Date (MM/DD/YYYY)	Priority Not Claimed	Certified Copy Attached? YES	NO
			[ ]	[ ]	[ ]
			[ ]	[ ]	[ ]
			[ ]	[ ]	[ ]
			[ ]	[ ]	[ ]

[ ] Additional foreign application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto:

Burden Hour Statement: This form is estimated to take 21 minutes to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

PTO/SB/01 (03-01)
Approved for use through 10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

DECLARATION — Utility or Design Patent Application

Direct all correspondence to: ☐ Customer Number or Bar Code Label		OR ☑ Correspondence address below	
Name: David B. Cochran, Esq.			
Address: JONES DAY North Point, 901 Lakeside Avenue			
City: Cleveland	State: Ohio	ZIP: 44114-1190	
Country: USA	Telephone: (216) 586-3939	Fax: (216) 579-0212	

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

NAME OF SOLE OR FIRST INVENTOR : ☐ A petition has been filed for this unsigned inventor

1-00

Given Name (first and middle [if any]): David P.	Family Name or Surname: YACH		
Inventor's Signature: David P. Yach CAX	Date: 17 March 2003		
Residence: City: Waterloo	State: Ontario	Country: CANADA	Citizenship: Canadian
Mailing Address: 295 Phillip Street			
City: Waterloo	State: Ontario	ZIP: N2L 3W8	Country: CANADA

NAME OF SECOND INVENTOR: ☐ A petition has been filed for this unsigned inventor

2-00

Given Name (first and middle [if any]): Michael S.	Family Name or Surname: BROWN		
Inventor's Signature: Michael S. Brown	Date: Mar 10, 2003		
Residence: City: Waterloo CAX	State: Ontario	Country: CANADA	Citizenship: Canadian
Mailing Address: 295 Phillip Street			
City: Waterloo	State: Ontario	ZIP: N2L 3W8	Country: CANADA

☑ Additional inventors are being named on the 1 supplemental Additional Inventor(s) sheet(s) PTO/SB/02A attached hereto.

PTO/SB/02A (10-00)
Approved for use through 10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

DECLARATION

ADDITIONAL INVENTOR(S)
Supplemental Sheet
Page 1 of 1

3.00

Name of Additional Joint Inventor, if any:	☐ A petition has been filed for this unsigned inventor		
Given Name: Herbert A.	Family Name or Surname: LITTLE		
Inventor's Signature: H A Little CAX			Date: Mar 12 2003.
Residence: City: Waterloo	State: Ontario	Country: CANADA	Citizenship: Canadian
Mailing Address: 295 Phillip Street			
Mailing Address:			
City: Waterloo	State: Ontario	ZIP: N2L 3W8	Country: CANADA

Name of Additional Joint Inventor, if any:	☐ A petition has been filed for this unsigned inventor		
Given Name:	Family Name or Surname:		
Inventor's Signature:			Date:
Residence: City:	State:	Country:	Citizenship:
Mailing Address:			
Mailing Address:			
City:	State:	ZIP:	Country:

Name of Additional Joint Inventor, if any:	☐ A petition has been filed for this unsigned inventor		
Given Name:	Family Name or Surname:		
Inventor's Signature:			Date:
Residence: City:	State:	Country:	Citizenship:
Mailing Address:			
Mailing Address:			
City:	State:	ZIP:	Country:

Burden Hour Statement: This form is estimated to take 21 minutes to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

Express Mail No. EV 243791125 US
March 20, 2003

Please type a plus sign (+) inside this box → [+]

PTO/SB/81 (02-01)
Approved for use through 10/31/2002. OMB 0651-0035
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it display a valid OMB control number.

POWER OF ATTORNEY OR AUTHORIZATION OF AGENT

Application Number	
Filing Date	
First Named Inventor	David P. YACH
Title	Software Code Signing ...
Group Art Unit	
Examiner Name	
Attorney Docket Number	555255012423

I hereby appoint:

☐ Practitioners at Customer Number [ ] → *Place Customer Number Bar Code Label here*

OR

☑ Practitioner(s) named below:

Name	Registration Number
Krishna K. Pathiyal, Esq.	44435
Robert C. Liang, Esq.	48091
Please see attached sheet	

as my/our attorney(s) or agent(s) to prosecute the application identified above, and to transact all business in the United States Patent and Trademark Office connected therewith.

Please change the correspondence address for the above-identified application to:

☐ The above-mentioned Customer Number.

OR

☐ Practitioners at Customer Number [ ] → *Place Customer Number Bar Code Label here*

OR

☑ Firm *or* Individual Name	**David B. Cochran, Esq.**				
Address	JONES DAY				
Address	North Point, 901 Lakeside Avenue				
City	Cleveland	State	Ohio	Zip	44114
Country	USA				
Telephone	(216) 586-3939	Fax	(216) 579-0212		

I am the:

☐ Applicant/Inventor.

☑ Assignee of record of the entire interest. See 37 CFR 3.71.
Statement under 37 CFR 3.73(b) is enclosed. (Form PTO/SB/96).

SIGNATURE of Applicant or Assignee of Record

Name	Mihal Lazaridis, President and Co-CEO, on behalf of Research In Motion Limited
Signature	
Date	7 Mar 03

NOTE: Signatures of all the inventors or assignees of record of the entire interest or their representative(s) are required. Submit multiple forms if more than one signature is required, see below*.

☑ *Total of 2 forms are submitted. (PTO/SB/81 (02-01) and *Supplemental Page Listing Additional Agents of Record)

Burden Hour Statement: This form is estimated to take 3 minutes to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

555255012423

SOFTWARE CODE SIGNING SYSTEM AND METHOD

* SUPPLEMENTAL PAGE LISTING ADDITIONAL AGENTS OF RECORD

ADAMO, Kenneth R., Reg. No. 27,299
ARNDT, Barbara E., Reg. No. 37,768
ASAM, Michael R. Reg. No. 51,417
BIERNACKI, John V., Reg. No. 40,511
COCHRAN, David B., Reg. No. 39,142
COOPER, Lorri W., Reg. No. 40,038
FAY, Regan J., Reg. No. 26,878
FEELING, F. Drexel, Reg. No. 40,602
FRANZ, Paul E., Reg. No. 45,910
GRIFFITH, Calvin P., Reg No. 34,831
MAIORANA, David M., Reg. No. 41,449
O'HEARN, Timothy J., Reg. No. 31,552
ROSE, Mitchell, Reg. No. 47,906
SAUER, Joseph M., Reg. No. 47,919
SCANLON, Stephen D., Reg. No. 32,755
SERRA, Wayne M., Reg. No. 51,138
SHEAFFER, Jenny L., Reg. No. 45,099
SWITZER, H. Duane, Reg. No. 22,431
VARY, Michael W., Reg. No. 30,811
WAMSLEY, III, James L., Reg. No. 31,578

all of JONES DAY
North Point
901 Lakeside Avenue
Cleveland, Ohio 44114
US

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

PTO-1556
(5/87)

*U.S. Government Printing Office: 2001 — 481-697/59173

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2001

Application or Docket Number

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS		
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	109 minus 20=	* 89
INDEPENDENT CLAIMS	12 minus 3 =	* 9
MULTIPLE DEPENDENT CLAIM PRESENT		☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐ OR OTHER THAN SMALL ENTITY

RATE	FEE		RATE	FEE
BASIC FEE		OR	BASIC FEE	900
X$ 9=		OR	X$18=	1602
X42=		OR	X84=	756
+140=		OR	+280=	
TOTAL		OR	TOTAL	

CLAIMS AS AMENDED - PART II

AMENDMENT A	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY OR OTHER THAN SMALL ENTITY

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than th ntry in column 2, write "0" in column 3.
** If the "High st Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, nter "3."
Th "Highest Number Previously Paid For" (Total or Independent) is the high st number found in the appropriate box in column 1.

FORM PTO-875 (Rev. 8/01)

☆U.S.GPO:2001 482-124 / 59197

Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE

TAMALA HOLLAND
PARALEGAL SPECIALIST
DESIGNATED OFFICE
(703) 305-5482

MULTIPLE DEPENDENT CLAIM FEE CALCULATION SHEET

(FOR USE WITH FORM PTO-875)

SERIAL NO. | FILING DATE

APPLICANT(S)

CLAIMS

	AS FILED		AFTER 1st AMENDMENT		AFTER 2nd AMENDMENT			*		*		*	
	IND.	DEP.	IND.	DEP.	IND.	DEP.		IND.	DEP.	IND.	DEP.	IND.	DEP.
1	/		/				51		/		/		
2		/		/			52		/		/		
3		/		/			53		/		/		
4		/		/			54		/		/		
5		/		/			55		/		/		
6			/				56		/	/			
7		/		/			57				/		
8		/		/			58				/		
9		/					59				/		
10		/					60				/		
11		/					61				/		
12		/		/			62				/		
13		/		/			63				/		
14		/		/			64				/		
15		/		/			65				/		
16		/		/			66				/		
17		/		/			67				/		
18		/		/			68			/			
19		/		/			69				/		
20		/		/			70				/		
21		/		/			71				/		
22		/		/			72				/		
23		/		/			73				/		
24		/		/			74				/		
25		/		/			75				/		
26		/		/			76				/		
27		/	/				77			/			
28		/		/			78				/		
29	/			/			79				/		
30		/		/			80				/		
31		/		/			81				/		
32		/		/			82				/		
33		/		/			83				/		
34		/		/			84				/		
35		/		/			85				/		
36		/	/				86				/		
37		/		/			87			/			
38	/			/			88				/		
39		/		/			89				/		
40		/		/			90				/		
41		/		/			91				/		
42		/		/			92				/		
43		/	/				93				/		
44		/		/			94				/		
45	/			/			95				/		
46		/		/			96			/			
47		/	/				97				/		
48		/		/			98				/		
49	/			/			99				/		
50		/		/			100				/		
TOTAL IND.							TOTAL IND.						
TOTAL DEP.							TOTAL DEP.						
TOTAL CLAIMS							TOTAL CLAIMS						

PTO-1360 (3-78)

*MAY BE USED FOR ADDITIONAL CLAIMS OR AMENDMENTS

U.S. DEPARTMENT of COMMERCE
Patent and Trademark Office

2 of 2

TAMALA HOLLAND
PARALEGAL SPECIALIST
DESIGNATED OFFICE
(703) 305-5483

MULTIPLE DEPENDENT CLAIM FEE CALCULATION SHEET
(FOR USE WITH FORM PTO-875)

SERIAL NO.

FILING DATE

APPLICANT(S)

CLAIMS

	AS FILED		AFTER 1st AMENDMENT		AFTER 2nd AMENDMENT			•		•		•	
	IND.	DEP.	IND.	DEP.	IND.	DEP.		IND.	DEP.	IND.	DEP.	IND.	DEP.
1				1			51						
2				1			52						
3				1			53						
4			1				54						
5				1			55						
6				1			56						
7				1			57						
8				1			58						
9				1			59						
10							60						
11							61						
12							62						
13							63						
14							64						
15							65						
16							66						
17							67						
18							68						
19							69						
20							70						
21							71						
22							72						
23							73						
24							74						
25							75						
26							76						
27							77						
28							78						
29							79						
30							80						
31							81						
32							82						
33							83						
34							84						
35							85						
36							86						
37							87						
38							88						
39							89						
40							90						
41							91						
42							92						
43							93						
44							94						
45							95						
46							96						
47							97						
48							98						
49							99						
50							100						
TOTAL IND.			12				TOTAL IND.						
TOTAL DEP.			97				TOTAL DEP.						
TOTAL CLAIMS			109				TOTAL CLAIMS						

PTO-1360 (3-78) *MAY BE USED FOR ADDITIONAL CLAIMS OR AMENDMENTS

U.S. DEPARTMENT of COMMERCE
Patent and Trademark Office

PATENT COOPERATION TREATY

PCT

INTERNATIONAL SEARCH REPORT

(PCT Article 18 and Rules 43 and 44)

Applicant's or agent's file reference	**FOR FURTHER ACTION** see Notification of Transmittal of International Search Report (Form PCT/ISA/220) as well as, where applicable, item 5 below.	
PCA-0445		
International application No.	International filing date *(day/month/year)*	(Earliest) Priority Date *(day/month/year)*
PCT/CA 01/ 01344	20/09/2001	21/09/2000
Applicant		
RESEARCH IN MOTION LIMITED		

This International Search Report has been prepared by this International Searching Authority and is transmitted to the applicant according to Article 18. A copy is being transmitted to the International Bureau.

This International Search Report consists of a total of 3 sheets.

[X] It is also accompanied by a copy of each prior art document cited in this report.

1. **Basis of the report**

   a. With regard to the **language,** the international search was carried out on the basis of the international application in the language in which it was filed, unless otherwise indicated under this item.

   [ ] the international search was carried out on the basis of a translation of the international application furnished to this Authority (Rule 23.1(b)).

   b. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, the international search was carried out on the basis of the sequence listing :

   [ ] contained in the international application in written form.

   [ ] filed together with the international application in computer readable form.

   [ ] furnished subsequently to this Authority in written form.

   [ ] furnished subsequently to this Authority in computer readble form.

   [ ] the statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.

   [ ] the statement that the information recorded in computer readable form is identical to the written sequence listing has been furnished

2. [ ] **Certain claims were found unsearchable** (See Box I).

3. [ ] **Unity of invention is lacking** (see Box II).

4. With regard to the **title,**

   [ ] the text is approved as submitted by the applicant.

   [X] the text has been established by this Authority to read as follows:

   SOFTWARE CODE SIGNING SYSTEM AND METHOD

5. With regard to the **abstract,**

   [X] the text is approved as submitted by the applicant.

   [ ] the text has been established, according to Rule 38.2(b), by this Authority as it appears in Box III. The applicant may, within one month from the date of mailing of this international search report, submit comments to this Authority.

6. The figure of the **drawings** to be published with the abstract is Figure No. 2

   [ ] as suggested by the applicant.

   [X] because the applicant failed to suggest a figure.

   [ ] because this figure better characterizes the invention.

   [ ] None of the figures.

Form PCT/ISA/210 (first sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

International Application No
[redacted]CA 01/01344

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 99 05600 A (APPLE COMPUTER) 4 February 1999 (1999-02-04) abstract; figures 5,6,9 page 6, line 1 - line 15 page 19, line 4 - line 14 page 20, line 19 -page 21, line 4 page 24, line 6 - line 23 page 25, line 23 - line 26	1,2,6,7, 12-15, 21,26, 27,29,32
Y	--- -/--	11,18, 19,26, 31,38-56

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
12 April 2002	22/04/2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Powell, D

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
[redacted]CA 01/01344

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 930 793 A (TEXAS INSTRUMENTS INC) 21 July 1999 (1999-07-21) abstract; figure 6 page 15, line 54 -page 16, line 5 page 16, line 32 - line 44	1,3-6, 8-10,20, 22-24, 28-33, 36,37
Y	---	34,35
P,Y	US 6 157 721 A (SIBERT W OLIN ET AL) 5 December 2000 (2000-12-05) abstract; figures 2,3,5,8,14 column 2, line 27 - line 65 column 11, line 7 - line 19 column 15, line 23 - line 41	11,18, 19,26, 31,34, 35,38-56
Y	& AU 36815 97 A (INTERTRUST TECHNOLOGIES CORP) 19 February 1998 (1998-02-19)	
A	--- US 5 978 484 A (APPERSON NORMAN ET AL) 2 November 1999 (1999-11-02) abstract; figure 5 column 2, line 41 - line 60 column 3, line 44 - line 57 column 8, line 17 - line 25	11,18, 19,31, 34,35

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/CA 01/01344

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9905600 A	04-02-1999	US 6188995 B1	13-02-2001
		EP 1023664 A2	02-08-2000
		WO 9905600 A2	04-02-1999
EP 0930793 A	21-07-1999	CN 1249643 A	05-04-2000
		EP 0930793 A1	21-07-1999
		JP 11312152 A	09-11-1999
US 6157721 A	05-12-2000	AU 3205797 A	05-12-1997
		AU 3681597 A	19-02-1998
		CN 1225739 A	11-08-1999
		EP 0898777 A2	03-03-1999
		JP 2001501763 T	06-02-2001
		WO 9743761 A2	20-11-1997
		US 6292569 B1	18-09-2001
		US 2002023214 A1	21-02-2002
US 5978484 A	02-11-1999	NONE	

Form PCT/ISA/210 (patent family annex) (July 1992)

PATENT COOPERATION TREAT

From the INTERNATIONAL BUREAU

PCT

NOTIFICATION OF ELECTION

(PCT Rule 61.2)

To:

Commissioner
US Department of Commerce
United States Patent and Trademark Office, PCT
2011 South Clark Place Room CP2/5C24
Arlington, VA 22202
United States of America

in its capacity as elected Office

Date of mailing (day/month/year) 17 September 2002 (17.09.02)	
International application No. PCT/CA01/01344	**Applicant's or agent's file reference** PCA-0445
International filing date (day/month/year) 20 September 2001 (20.09.01)	**Priority date** (day/month/year) 21 September 2000 (21.09.00)
Applicant YACH, David, P. et al	

1. The designated Office is hereby notified of its election made:

[X] in the demand filed with the International Preliminary Examining Authority on:

22 April 2002 (22.04.02)

[ ] in a notice effecting later election filed with the International Bureau on:

2. The election [X] was

[ ] was not

made before the expiration of 19 months from the priority date or, where Rule 32 applies, within the time limit under Rule 32.2(b).

The International Bureau of WIPO **34, chemin des Colombett s** **1211 Geneva 20, Switzerland**	Authorized officer Denise POSPIEZNY
Facsimile No.: (41-22) 740.14.35	Telephone No.: (41-22) 338.83.38

Form PCT/IB/331 (July 1992) CA0101344

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
28 March 2002 (28.03.2002)

PCT

(10) International Publication Number
WO 02/25409 A3

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/CA01/01344

(22) International Filing Date:
20 September 2001 (20.09.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/234,152	21 September 2000 (21.09.2000)	US
60/235,354	26 September 2000 (26.09.2000)	US
60/270,663	20 February 2001 (20.02.2001)	US

(71) Applicant *(for all designated States except US)*: **RESEARCH IN MOTION LIMITED** [CA/CA]; 295 Phillip Street, Waterloo, Ontario N2L 3W8 (CA).

(72) Inventors; and

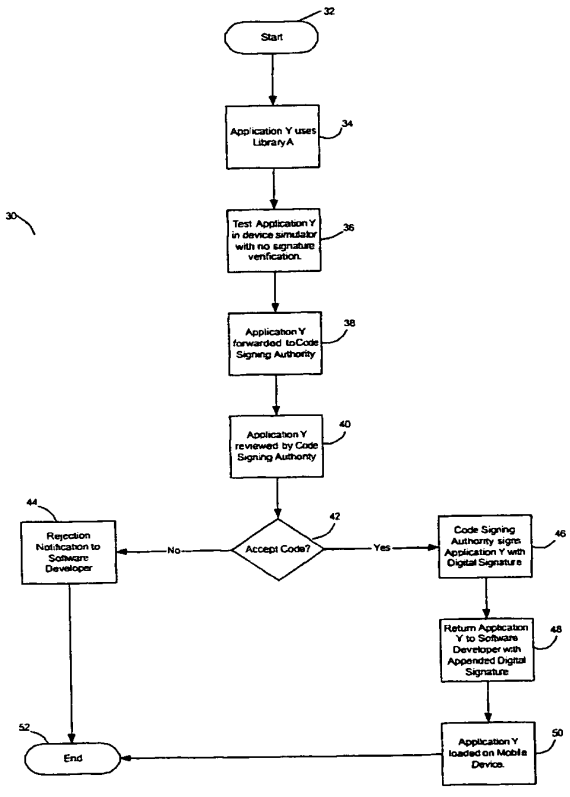
(75) Inventors/Applicants *(for US only)*: **YACH, David, P.** [CA/CA]; 254 Castlefield Avenue, Waterloo, Ontario N2K 2N1 (CA). **BROWN, Michael, S.** [CA/CA]; 7 Danube Street, Heidelberg, Ontario N0B 1Y0 (CA). **LITTLE, Herbert, A.** [CA/CA]; 504 Old Oak Place, Waterloo, Ontario N2T 2V8 (CA).

(74) Agent: **PATHIYAL, Krishna, K.**; Research In Motion Limited, 295 Phillip Street, Waterloo, Ontario N2L 3W8 (CA).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

[Continued on next page]

(54) Title: SOFTWARE CODE SIGNING SYSTEM AND METHOD



(57) Abstract: A code signing system and method is provided. The code signing system operates in conjunction with a signed software application having a digital signature and includes an application platform, an application programming interface (API), and a virtual machine. The API is configured to link the software application with the application platform. The virtual machine verifies the authenticity of the digital signature in order to control access to the API by the software application.

WO 02/25409 A3

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

(88) **Date of publication of the international search report:**
13 June 2002

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

PATENT COOPERATION TREATY

PCT

INTERNATIONAL SEARCH REPORT

(PCT Article 18 and Rules 43 and 44)

Applicant's or agent's file reference	FOR FURTHER ACTION	see Notification of Transmittal of International Search Report (Form PCT/ISA/220) as well as, where applicable, item 5 below.
PCA-0445		
International application No.	International filing date *(day/month/year)*	(Earliest) Priority Date *(day/month/year)*
PCT/CA 01/ 01344	20/09/2001	21/09/2000

Applicant

RESEARCH IN MOTION LIMITED

This International Search Report has been prepared by this International Searching Authority and is transmitted to the applicant according to Article 18. A copy is being transmitted to the International Bureau.

This International Search Report consists of a total of 3 sheets.

[X] It is also accompanied by a copy of each prior art document cited in this report.

1. **Basis of the report**

a. With regard to the **language**, the international search was carried out on the basis of the international application in the language in which it was filed, unless otherwise indicated under this item.

[ ] the international search was carried out on the basis of a translation of the international application furnished to this Authority (Rule 23.1(b)).

b. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, the international search was carried out on the basis of the sequence listing :

[ ] contained in the international application in written form.

[ ] filed together with the international application in computer readable form.

[ ] furnished subsequently to this Authority in written form.

[ ] furnished subsequently to this Authority in computer readble form.

[ ] the statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.

[ ] the statement that the information recorded in computer readable form is identical to the written sequence listing has been furnished

2. [ ] **Certain claims were found unsearchable** (See Box I).

3. [ ] **Unity of invention is lacking** (see Box II).

4. With regard to the **title,**

[ ] the text is approved as submitted by the applicant.

[X] the text has been established by this Authority to read as follows:

SOFTWARE CODE SIGNING SYSTEM AND METHOD

5. With regard to the **abstract,**

[X] the text is approved as submitted by the applicant.

[ ] the text has been established, according to Rule 38.2(b), by this Authority as it appears in Box III. The applicant may, within one month from the date of mailing of this international search report, submit comments to this Authority.

6. The figure of the **drawings** to be published with the abstract is Figure No. 2

[ ] as suggested by the applicant.

[X] because the applicant failed to suggest a figure.

[ ] because this figure better characterizes the invention.

[ ] None of the figures.

Form PCT/ISA/210 (first sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

national Application No
T/CA 01/01344

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 99 05600 A (APPLE COMPUTER) 4 February 1999 (1999-02-04) abstract; figures 5,6,9 page 6, line 1 - line 15 page 19, line 4 - line 14 page 20, line 19 -page 21, line 4 page 24, line 6 - line 23 page 25, line 23 - line 26	1,2,6,7, 12-15, 21,26, 27,29,32
Y		11,18, 19,26, 31,38-56

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
12 April 2002	22/04/2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Powell, D

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

national Application No
T/CA 01/01344

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 930 793 A (TEXAS INSTRUMENTS INC) 21 July 1999 (1999-07-21) abstract; figure 6 page 15, line 54 -page 16, line 5 page 16, line 32 - line 44	1,3-6, 8-10,20, 22-24, 28-33, 36,37
Y	---	34,35
P,Y	US 6 157 721 A (SIBERT W OLIN ET AL) 5 December 2000 (2000-12-05) abstract; figures 2,3,5,8,14 column 2, line 27 - line 65 column 11, line 7 - line 19 column 15, line 23 - line 41	11,18, 19,26, 31,34, 35,38-56
Y	& AU 36815 97 A (INTERTRUST TECHNOLOGIES CORP) 19 February 1998 (1998-02-19)	

A	US 5 978 484 A (APPERSON NORMAN ET AL) 2 November 1999 (1999-11-02) abstract; figure 5 column 2, line 41 - line 60 column 3, line 44 - line 57 column 8, line 17 - line 25	11,18, 19,31, 34,35

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

national Application No
T/CA 01/01344

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
WO 9905600	A	04-02-1999	US	6188995 B1	13-02-2001
			EP	1023664 A2	02-08-2000
			WO	9905600 A2	04-02-1999
EP 0930793	A	21-07-1999	CN	1249643 A	05-04-2000
			EP	0930793 A1	21-07-1999
			JP	11312152 A	09-11-1999
US 6157721	A	05-12-2000	AU	3205797 A	05-12-1997
			AU	3681597 A	19-02-1998
			CN	1225739 A	11-08-1999
			EP	0898777 A2	03-03-1999
			JP	2001501763 T	06-02-2001
			WO	9743761 A2	20-11-1997
			US	6292569 B1	18-09-2001
			US	2002023214 A1	21-02-2002
US 5978484	A	02-11-1999	NONE		

Form PCT/ISA/210 (patent family annex) (July 1992)

SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

7/PRTS

Express Mail No. EV 243791125 US
March 20, 2003
10/381219
DT09 Rec'd PCT/PTO 20 MAR 2003

SOFTWARE CODE SIGNING SYSTEM AND METHOD

EV243791125US

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority from and is related to the following prior applications: "Code Signing System And Method," United States Provisional Application No. 60/234,152, filed September 21, 2000; "Code Signing System And Method," United States Provisional Application No. 60/235,354, filed September 26, 2000; and "Code Signing System And Method," United States Provisional Application No. 60/270,663, filed February 20, 2001.

BACKGROUND

1. FIELD OF THE INVENTION

This invention relates generally to the field of security protocols for software applications. More particularly, the invention provides a code signing system and method that is particularly well suited for Java™ applications for mobile communication devices, such as Personal Digital Assistants, cellular telephones, and wireless two-way communication devices (collectively referred to hereinafter as "mobile devices" or simply "devices").

2. DESCRIPTION OF THE RELATED ART

Security protocols involving software code signing schemes are known. Typically, such security protocols are used to ensure the reliability of software applications that are downloaded from the Internet. In a typical software code signing scheme, a digital signature is attached to a software application that identifies the software developer. Once the software is downloaded by a user, the user typically must use his or her judgment to determine whether or not the software

application is reliable, based solely on his or her knowledge of the software developer's reputation. This type of code signing scheme does not ensure that a software application written by a third party for a mobile device will properly interact with the device's native applications and other resources. Because typical code signing protocols are not secure and rely solely on the judgment of the user, there is a serious risk that destructive, "Trojan horse" type software applications may be downloaded and installed onto a mobile device.

There also remains a need for network operators to have a system and method to maintain control over which software applications are activated on mobile devices.

There remains a further need in 2.5G and 3G networks where corporate clients or network operators would like to control the types of software on the devices issued to its employees.

SUMMARY

A code signing system and method is provided. The code signing system operates in conjunction with a software application having a digital signature and includes an application platform, an application programming interface (API), and a virtual machine. The API is configured to link the software application with the application platform. The virtual machine verifies the authenticity of the digital signature in order to control access to the API by the software application.

A code signing system for operation in conjunction with a software application having a digital signature, according to another embodiment of the invention comprises an application platform, a plurality of APIs, each configured to link the software application with a resource on

the application platform, and a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application, wherein the virtual machine verifies the authenticity of the digital signature in order to control access to the plurality of APIs by the software application.

According to a further embodiment of the invention, a method of controlling access to sensitive application programming interfaces on a mobile device comprises the steps of loading a software application on the mobile device that requires access to a sensitive API, determining whether or not the software application includes a digital signature associated with the sensitive API, and if the software application does not include a digital signature associated with the sensitive API, then denying the software application access to the sensitive API.

In another embodiment of the invention, a method of controlling access to an application programming interface (API) on a mobile device by a software application created by a software developer comprises the steps of receiving the software application from the software developer, reviewing the software application to determine if it may access the API, if the software application may access the API, then appending a digital signature to the software application, verifying the authenticity of a digital signature appended to a software application, and providing access to the API to software applications for which the appended digital signature is authentic.

A method of restricting access to a sensitive API on a mobile device, according to a further embodiment of the invention, comprises the steps of registering one or more software developers that are trusted to design software applications which access the sensitive API, receiving a hash of a software application, determining if the software application was designed by one of the registered software developers, and if the software application was designed by one

of the registered software developers, then generating a digital signature using the hash of the software application, wherein the digital signature may be appended to the software application, and the mobile device verifies the authenticity of the digital signature in order to control access to the sensitive API by the software application.

In a still further embodiment, a method of restricting access to application programming interfaces on a mobile device comprises the steps of loading a software application on the mobile device that requires access to one or more API, determining whether or not the software application includes a digital signature associated with the mobile device, and if the software application does not include a digital signature associated with the mobile device, then denying the software application access to the one or more APIs.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram illustrating a code signing protocol according to one embodiment of the invention;

Fig. 2 is a flow diagram of the code signing protocol described above with reference to Fig. 1;

Fig. 3 is a block diagram of a code signing system on a mobile device;

Fig. 3A is a block diagram of a code signing system on a plurality of mobile devices;

Fig. 4 is a flow diagram illustrating the operation of the code signing system described above with reference to Fig. 3 and Fig. 3A;

Fig. 5 is a flow diagram illustrating the management of the code signing authorities described with reference to Fig. 3A; and

Fig. 6 is a block diagram of a mobile communication device in which a code signing system and method may be implemented.

DETAILED DESCRIPTION

Referring now to the drawing figures, Fig. 1 is a diagram illustrating a code signing protocol according to one embodiment of the invention. An application developer 12 creates a software application 14 (application Y) for a mobile device that requires access to one or more sensitive APIs on the mobile device. The software application Y 14 may, for example, be a Java application that operates on a Java virtual machine installed on the mobile device. An API enables the software application Y to interface with an application platform that may include, for example, resources such as the device hardware, operating system and core software and data models. In order to make function calls to or otherwise interact with such device resources, a software application Y must access one or more APIs. APIs can thereby effectively "bridge" a software application and associated device resources. In this description and the appended claims, references to API access should be interpreted to include access of an API in such a way as to allow a software application Y to interact with one or more corresponding device resources. Providing access to any API therefore allows a software application Y to interact with associated device resources, whereas denying access to an API prevents the software application Y from interacting with the associated resources. For example, a database API may communicate with a device file or data storage system, and access to the database API would provide for interaction between a software application Y and the file or data storage system. A user interface (UI) API would communicate with controllers and/or control software for such device components as a

screen, a keyboard, and any other device components that provide output to a user or accept input from a user. In a mobile device, a radio API may also be provided as an interface to wireless communication resources such as a transmitter and receiver. Similarly, a cryptographic API may be provided to interact with a crypto module which implements crypto algorithms on a device. These are merely illustrative examples of APIs that may be provided on a device. A device may include any of these example APIs, or different APIs instead of or in addition to those described above.

Preferably, any API may be classified as sensitive by a mobile device manufacturer, or possibly by an API author, a wireless network operator, a device owner or operator, or some other entity that may be affected by a virus or malicious code in a device software application. For instance, a mobile device manufacturer may classify as sensitive those APIs that interface with cryptographic routines, wireless communication functions, or proprietary data models such as address book or calendar entries. To protect against unauthorized access to these sensitive APIs, the application developer 12 is required to obtain one or more digital signatures from the mobile device manufacturer or other entity that classified any APIs as sensitive, or from a code signing authority 16 acting on behalf of the manufacturer or other entity with an interest in protecting access to sensitive device APIs, and append the signature(s) to the software application Y 14.

In one embodiment, a digital signature is obtained for each sensitive API or library that includes a sensitive API to which the software application requires access. In some cases, multiple signatures are desirable. This would allow a service provider, company or network operator to restrict some or all software applications loaded or updated onto a particular set of

mobile devices. In this multiple-signature scenario, all APIs are restricted and locked until a "global" signature is verified for a software application. For example, a company may wish to prevent its employees from executing any software applications onto their devices without first obtaining permission from a corporate information technology (IT) or computer services department. All such corporate mobile devices may then be configured to require verification of at least a global signature before a software application can be executed. Access to sensitive device APIs and libraries, if any, could then be further restricted, dependent upon verification of respective corresponding digital signatures.

The binary executable representation of software application Y 14 may be independent of the particular type of mobile device or model of a mobile device. Software application Y 14 may for example be in a write-once-run-anywhere binary format such as is the case with Java software applications. However, it may be desirable to have a digital signature for each mobile device type or model, or alternatively for each mobile device platform or manufacturer. Therefore, software application Y 14 may be submitted to several code signing authorities if software application Y 14 targets several mobile devices.

Software application Y 14 is sent from the application developer 12 to the code signing authority 16. In the embodiment shown in Fig. 1, the code signing authority 16 reviews the software application Y 14, although as described in further detail below, it is contemplated that the code signing authority 16 may also or instead consider the identity of the application developer 12 to determine whether or not the software application Y 14 should be signed. The code signing authority 16 is preferably one or more representatives from the mobile device

manufacturer, the authors of any sensitive APIs, or possibly others that have knowledge of the operation of the sensitive APIs to which the software application needs access.

If the code signing authority 16 determines that software application Y 14 may access the sensitive API and therefore should be signed, then a signature (not shown) for the software application Y 14 is generated by the code signing authority 16 and appended to the software application Y 14. The signed software application Y 22, comprising the software application Y 14 and the digital signature, is then returned to the application developer 12. The digital signature is preferably a tag that is generated using a private signature key 18 maintained solely by the code signing authority 16. For example, according to one signature scheme, a hash of the software application Y 14 may be generated, using a hashing algorithm such as the Secure Hash Algorithm SHA1, and then used with the private signature key 18 to create the digital signature. In some signature schemes, the private signature key is used to encrypt a hash of information to be signed, such as software application Y 14, whereas in other schemes, the private key may be used in other ways to generate a signature from the information to be signed or a transformed version of the information.

The signed software application Y 22 may then be sent to a mobile device 28 or downloaded by the mobile device 28 over a wireless network 24. It should be understood, however, that a code signing protocol according to the present invention is not limited to software applications that are downloaded over a wireless network. For instance, in alternative embodiments, the signed software application Y 22 may be downloaded to a personal computer via a computer network and loaded to the mobile device through a serial link, or may be acquired from the application developer 12 in any other manner and loaded onto the mobile device. Once

the signed software application Y 22 is loaded on the mobile device 28, each digital signature is preferably verified with a public signature key 20 before the software application Y 14 is granted access to a sensitive API library. Although the signed software application Y 22 is loaded onto a device, it should be appreciated that the software application that may eventually be executed on the device is the software application Y 14. As described above, the signed software application Y 22 includes the software application Y 14 and one or more appended digital signatures (not shown). When the signatures are verified, the software application Y 14 can be executed on the device and access any APIs for which corresponding signatures have been verified.

The public signature key 20 corresponds to the private signature key 18 maintained by the code signing authority 16, and is preferably installed on the mobile device along with the sensitive API. However, the public key 10 may instead be obtained from a public key repository (not shown), using the device 28 or possibly a personal computer system, and installed on the device 28 as needed. According to one embodiment of a signature scheme, the mobile device 28 calculates a hash of the software application Y 14 in the signed software application Y 22, using the same hashing algorithm as the code signing authority 16, and uses the digital signature and the public signature key 20 to recover the hash calculated by the signing authority 16. The resultant locally calculated hash and the hash recovered from the digital signature are then compared, and if the hashes are the same, the signature is verified. The software application Y 14 can then be executed on the device 28 and access any sensitive APIs for which the corresponding signature(s) have been verified. As described above, the invention is in no way limited to this particular illustrative example signature scheme. Other signature schemes,

including further public key signature schemes, may also be used in conjunction with the code signing methods and systems described herein.

Fig. 2 is a flow diagram 30 of the code signing protocol described above with reference to Fig. 1. The protocol begins at step 32. At step 34, a software developer writes the software application Y for a mobile device that requires access to a sensitive API or library that exposes a sensitive API (API library A). As discussed above, some or all APIs on a mobile device may be classified as sensitive, thus requiring verification of a digital signature for access by any software application such as software application Y. In step 36, application Y is tested by the software developer, preferably using a device simulator in which the digital signature verification function has been disabled. In this manner, the software developer may debug the software application Y before the digital signature is acquired from the code signing authority. Once the software application Y has been written and debugged, it is forwarded to the code signing authority in step 38.

In steps 40 and 42, the code signing authority reviews the software application Y to determine whether or not it should be given access to the sensitive API, and either accepts or rejects the software application. The code signing authority may apply a number of criteria to determine whether or not to grant the software application access to the sensitive API including, for example, the size of the software application, the device resources accessed by the API, the perceived utility of the software application, the interaction with other software applications, the inclusion of a virus or other destructive code, and whether or not the developer has a contractual obligation or other business arrangement with the mobile device manufacturer. Further details of managing code signing authorities and developers are described below in reference to Fig. 5.

If the code signing authority accepts the software application Y, then a digital signature, and preferably a signature identification, are appended to the software application Y in step 46. As described above, the digital signature may be generated by using a hash of the software application Y and a private signature key 18. The signature identification is described below with reference to Figs. 3 and 4. Once the digital signature and signature identification are appended to the software application Y to generate a signed software application, the signed software application Y is returned to the software developer in step 48. The software developer may then license the signed software application Y to be loaded onto a mobile device (step 50). If the code signing authority rejects the software application Y, however, then a rejection notification is preferably sent to the software developer (step 44), and the software application Y will be unable to access any API(s) associated with the signature.

In an alternative embodiment, the software developer may provide the code signing authority with only a hash of the software application Y, or provide the software application Y in some type of abridged format. If the software application Y is a Java application, then the device independent binary *.class files may be used in the hashing operation, although device dependent files such as *.cod files used by the assignee of the present application may instead be used in hashing or other digital signature operations when software applications are intended for operation on particular devices or device types. By providing only a hash or abridged version of the software application Y, the software developer may have the software application Y signed without revealing proprietary code to the code signing authority. The hash of the software application Y, along with the private signature key 18, may then be used by the code signing authority to generate the digital signature. If an otherwise abridged version of the software

application Y is sent to the code signing authority, then the abridged version may similarly be used to generate the digital signature, provided that the abridging scheme or algorithm, like a hashing algorithm, generates different outputs for different inputs. This ensures that every software application will have a different abridged version and thus a different signature that can only be verified when appended to the particular corresponding software application from which the abridged version was generated. Because this embodiment does not enable the code signing authority to thoroughly review the software application for viruses or other destructive code, however, a registration process between the software developer and the code signing authority may also be required. For instance, the code signing authority may agree in advance to provide a trusted software developer access to a limited set of sensitive APIs.

In still another alternative embodiment, a software application Y may be submitted to more than one signing authority. Each signing authority may for example be responsible for signing software applications for particular sensitive APIs or APIs on a particular model of mobile device or set of mobile devices that supports the sensitive APIs required by a software application. A manufacturer, mobile communication network operator, service provider, or corporate client for example may thereby have signing authority over the use of sensitive APIs for their particular mobile device model(s), or the mobile devices operating on a particular network, subscribing to one or more particular services, or distributed to corporate employees. A signed software application may then include a software application and at least one appended digital signature appended from each of the signing authorities. Even though these signing authorities in this example would be generating a signature for the same software application,

different signing and signature verification schemes may be associated with the different signing authorities.

Fig. 3 is a block diagram of a code signing system 60 on a mobile device 62. The system 60 includes a virtual machine 64, a plurality of software applications 66-70, a plurality of API libraries 72-78, and an application platform 80. The application platform 80 preferably includes all of the resources on the mobile device 62 that may be accessed by the software applications 66-70. For instance, the application platform may include device hardware 82, the mobile device's operating system 84, or core software and data models 86. Each API library 72-78 preferably includes a plurality of APIs that interface with a resource available in the application platform. For instance, one API library might include all of the APIs that interface with a calendar program and calendar entry data models. Another API library might include all of the APIs that interface with the transmission circuitry and functions of the mobile device 62. Yet another API library might include all of the APIs capable of interfacing with lower-level services performed by the mobile device's operating system 84. In addition, the plurality of API libraries 72-78 may include both libraries that expose a sensitive API 74 and 78, such as an interface to a cryptographic function, and libraries 72 and 76, that may be accessed without exposing sensitive APIs. Similarly, the plurality of software applications 66-70 may include both signed software applications 66 and 70 that require access to one or more sensitive APIs, and unsigned software applications such as 68. The virtual machine 64 is preferably an object oriented run-time environment such as Sun Micro System's J2ME™ (Java 2 Platform, Micro Edition), which manages the execution of all of the software applications 66-70 operating on the mobile device 62, and links the software applications 66-70 to the various API libraries 72-78.

Software application Y 70 is an example of a signed software application. Each signed software application preferably includes an actual software application such as software application Y comprising for example software code that can be executed on the application platform 80, one or more signature identifications 94 and one or more corresponding digital signatures 96. Preferably each digital signature 96 and associated signature identification 94 in a signed software application 66 or 70 corresponds to a sensitive API library 74 or 78 to which the software application X or software application Y requires access. The sensitive API library 74 or 78 may include one or more sensitive APIs. In an alternative embodiment, the signed software applications 66 and 70 may include a digital signature 96 for each sensitive API within an API library 74 or 78. The signature identifications 94 may be unique integers or some other means of relating a digital signature 96 to a specific API library 74 or 78, API, application platform 80, or model of mobile device 62.

API library A 78 is an example of an API library that exposes a sensitive API. Each API library 74 and 78 including a sensitive API should preferably include a description string 88, a public signature key 20, and a signature identifier 92. The signature identifier 92 preferably corresponds to a signature identification 94 in a signed software application 66 or 70, and enables the virtual machine 64 to quickly match a digital signature 96 with an API library 74 or 78. The public signature key 20 corresponds to the private signature key 18 maintained by the code signing authority, and is used to verify the authenticity of a digital signature 96. The description string 88 may for example be a textual message that is displayed on the mobile device when a signed software application 66 or 70 is loaded, or alternatively when a software application X or Y attempts to access a sensitive API.

Operationally, when a signed software application 68-70, respectively including a software application X, Z, or Y, that requires access to a sensitive API library 74 or 78 is loaded onto a mobile device, the virtual machine 64 searches the signed for an appended digital signature 96 associated with the API library 74 or 78. Preferably, the appropriate digital signature 96 is located by the virtual machine 64 by matching the signature identifier 92 in the API library 74 or 78 with a signature identification 94 on the signed software application. If the signed software application includes the appropriate digital signature 96, then the virtual machine 64 verifies its authenticity using the public signature key 20. Then, once the appropriate digital signature 96 has been located and verified, the description string 88 is preferably displayed on the mobile device before the software application X or Y is executed and accesses the sensitive API. For instance, the description string 88 may display a message stating that "Application Y is attempting to access API Library A," and thereby provide the mobile device user with the final control to grant or deny access to the sensitive API.

Fig. 3A is a block diagram of a code signing system 61 on a plurality of mobile devices 62E, 62F and 62G. The system 61 includes a plurality of mobile devices each of which only three are illustrated, mobile devices 62E, 62F and 62G. Also shown is a signed software application 70, including a software application Y to which two digital signatures 96E and 96F with corresponding signature identifications 94E and 94F have been appended. In the example system 61, each pair composed of a digital signature and identification, 94E/96E and 94F/96F, corresponds to a model of mobile device 62, API library 78, or associated platform 80. If signature identifications 94E and 94F correspond to different models of mobile device 62, then when a signed software application 70 which includes a software application Y that requires

access to a sensitive API library 78 is loaded onto mobile device 62E, the virtual machine 64 searches the signed software application 70 for a digital signature 96E associated with the API library 78 by matching identifier 94E with signature identifier 92. Similarly, when a signed software application 70 including a software application Y that requires access to a sensitive API library 78 is loaded onto a mobile device 62F, the virtual machine 64 in device 62F searches the signed software application 70 for a digital signature 96F associated with the API library 78. However, when a software application Y in a signed software application 70 that requires access to a sensitive API library 78 is loaded onto a mobile device model for which the application developer has not obtained a digital signature, device 62G in the example of Fig. 3A, the virtual machine 64 in the device 64G does not find a digital signature appended to the software application Y and consequently, access to the API library 78 is denied on device 62G. It should be appreciated from the foregoing description that a software application such as software application Y may have multiple device-specific, library-specific, or API-specific signatures or some combination of such signatures appended thereto. Similarly, different signature verification requirements may be configured for the different devices. For example, device 62E may require verification of both a global signature, as well as additional signatures for any sensitive APIs to which a software application requires access in order for the software application to be executed, whereas device 62F may require verification of only a global signature and device 62G may require verification of signatures only for its sensitive APIs. It should also be apparent that a communication system may include devices (not shown) on which a software application Y received as part of a signed software application such as 70 may execute without any signature verification. Although a signed software application has one or

more signatures appended thereto, the software application Y might possibly be executed on some devices without first having any of its signature(s) verified. Signing of a software application preferably does not interfere with its execution on devices in which digital signature verification is not implemented.

Fig. 4 is a flow diagram 100 illustrating the operation of the code signing system described above with reference to Figs. 3 and 3A. In step 102, a software application is loaded onto a mobile device. Once the software application is loaded, the device, preferably using a virtual machine, determines whether or not the software application requires access to any API libraries that expose a sensitive API (step 104). If not, then the software application is linked with all of its required API libraries and executed (step 118). If the software application does require access to a sensitive API, however, then the virtual machine verifies that the software application includes a valid digital signature associated any sensitive APIs to which access is required, in steps 106-116.

In step 106, the virtual machine retrieves the public signature key 20 and signature identifier 92 from the sensitive API library. The signature identifier 92 is then used by the virtual machine in step 108 to determine whether or not the software application has an appended digital signature 96 with a corresponding signature identification 94. If not, then the software application has not been approved for access to the sensitive API by a code signing authority, and the software application is preferably prevented from being executed in step 116. In alternative embodiments, a software application without a proper digital signature 96 may be purged from the mobile device, or may be denied access to the API library exposing the sensitive API but executed to the extent possible without access to the API library. It is also contemplated

that a user may be prompted for an input when signature verification fails, thereby providing for user control of such subsequent operations as purging of the software application from the device.

If a digital signature 96 corresponding to the sensitive API library is appended to the software application and located by the virtual machine, then the virtual machine uses the public key 20 to verify the authenticity of the digital signature 96 in step 110. This step may be performed, for example, by using the signature verification scheme described above or other alternative signature schemes. If the digital signature 96 is not authentic, then the software application is preferably either not executed, purged, or restricted from accessing the sensitive API as described above with reference to step 116. If the digital signature is authentic, however, then the description string 88 is preferably displayed in step 112, warning the mobile device user that the software application requires access to a sensitive API, and possibly prompting the user for authorization to execute or load the software application (step 114). When more than one signature is to be verified for a software application, then the steps 104-110 are preferably repeated for each signature before the user is prompted in step 112. If the mobile device user in step 114 authorizes the software application, then it may be executed and linked to the sensitive API library in step 118.

Fig. 5 is a flow diagram 200 illustrating the management of the code signing authorities described with reference to Fig. 3A. At step 210, an application developer has developed a new software application which is intended to be executable one or more target device models or types. The target devices may include sets of devices from different manufacturers, sets of device models or types from the same manufacturer, or generally any sets of devices having

particular signature and verification requirements. The term "target device" refers to any such set of devices having a common signature requirement. For example, a set of devices requiring verification of a device-specific global signature for execution of all software applications may comprise a target device, and devices that require both a global signature and further signatures for sensitive APIs may be part of more than one target device set. The software application may be written in a device independent manner by using at least one known API, supported on at least one target device with an API library. Preferably, the developed software application is intended to be executable on several target devices, each of which has its own at least one API library.

At step 220, a code signing authority for one target device receives a target-signing request from the developer. The target signing request includes the software application or a hash of the software application, a developer identifier, as well as at least one target device identifier which identifies the target device for which a signature is being requested. At step 230, the signing authority consults a developer database 235 or other records to determine whether or not to trust developer 220. This determination can be made according to several criteria discussed above, such as whether or not the developer has a contractual obligation or has entered into some other type of business arrangement with a device manufacturer, network operator, service provider, or device manufacturer. If the developer is trusted, then the method proceeds at step 240. However, if the developer is not trusted, then the software application is rejected (250) and not signed by the signing authority. Assuming the developer was trusted, at step 240 the signing authority determines if it has the target private key corresponding to the submitted target identifier by consulting a private key store such as a target private key database 245. If the target private key is found, then a digital signature for the software application is generated at step 260

and the digital signature or a signed software application including the digital signature appended to the software application is returned to the developer at step 280. However, if the target private key is not found at step 240, then the software application is rejected at step 270 and no digital signature is generated for the software application.

Advantageously, if target signing authorities follow compatible embodiments of the method outlined in Fig. 5, a network of target signing authorities may be established in order to expediently manage code signing authorities and a developer community code signing process providing signed software applications for multiple targets with low likelihood of destructive code.

Should any destructive or otherwise problematic code be found in a software application or suspected because of behavior exhibited when a software application is executed on a device, then the registration or privileges of the corresponding application developer with any or all signing authorities may also be suspended or revoked, since the digital signature provides an audit trail through which the developer of a problematic software application may be identified. In such an event, devices may be informed of the revocation by being configured to periodically download signature revocation lists, for example. If software applications for which the corresponding digital signatures have been revoked are running on a device, the device may then halt execution of any such software application and possibly purge the software application from its local storage. If preferred, devices may also be configured to re-execute digital signature verifications, for instance periodically or when a new revocation list is downloaded.

Although a digital signature generated by a signing authority is dependent upon authentication of the application developer and confirmation that the application developer has

been properly registered, the digital signature is preferably generated from a hash or otherwise transformed version of the software application and is therefore application-specific. This contrasts with known code signing schemes, in which API access is granted to any software applications arriving from trusted application developers or authors. In the code signing systems and methods described herein, API access is granted on an application-by-application basis and thus can be more strictly controlled or regulated.

Fig. 6 is a block diagram of a mobile communication device in which a code signing system and method may be implemented. The mobile communication device 610 is preferably a two-way communication device having at least voice and data communication capabilities. The device preferably has the capability to communicate with other computer systems on the Internet. Depending on the functionality provided by the device, the device may be referred to as a data messaging device, a two-way pager, a cellular telephone with data messaging capabilities, a wireless Internet appliance or a data communication device (with or without telephony capabilities).

Where the device 610 is enabled for two-way communications, the device will incorporate a communication subsystem 611, including a receiver 612, a transmitter 614, and associated components such as one or more, preferably embedded or internal, antenna elements 616 and 618, local oscillators (LOs) 613, and a processing module such as a digital signal processor (DSP) 620. As will be apparent to those skilled in the field of communications, the particular design of the communication subsystem 611 will be dependent upon the communication network in which the device is intended to operate. For example, a device 610 destined for a North American market may include a communication subsystem 611 designed to

operate within the Mobitex™ mobile communication system or DataTAC™ mobile communication system, whereas a device 610 intended for use in Europe may incorporate a General Packet Radio Service (GPRS) communication subsystem 611.

Network access requirements will also vary depending upon the type of network 919. For example, in the Mobitex and DataTAC networks, mobile devices such as 610 are registered on the network using a unique identification number associated with each device. In GPRS networks however, network access is associated with a subscriber or user of a device 610. A GPRS device therefore requires a subscriber identity module (not shown), commonly referred to as a SIM card, in order to operate on a GPRS network. Without a SIM card, a GPRS device will not be fully functional. Local or non-network communication functions (if any) may be operable, but the device 610 will be unable to carry out any functions involving communications over network 619, other than any legally required operations such as "911" emergency calling.

When required network registration or activation procedures have been completed, a device 610 may send and receive communication signals over the network 619. Signals received by the antenna 616 through a communication network 619 are input to the receiver 612, which may perform such common receiver functions as signal amplification, frequency down conversion, filtering, channel selection and the like, and in the example system shown in Fig. 6, analog to digital conversion. Analog to digital conversion of a received signal allows more complex communication functions such as demodulation and decoding to be performed in the DSP 620. In a similar manner, signals to be transmitted are processed, including modulation and encoding for example, by the DSP 620 and input to the transmitter 614 for digital to analog

conversion, frequency up conversion, filtering, amplification and transmission over the communication network 619 via the antenna 618.

The DSP 620 not only processes communication signals, but also provides for receiver and transmitter control. For example, the gains applied to communication signals in the receiver 612 and transmitter 614 may be adaptively controlled through automatic gain control algorithms implemented in the DSP 620.

The device 610 preferably includes a microprocessor 638 which controls the overall operation of the device. Communication functions, including at least data and voice communications, are performed through the communication subsystem 611. The microprocessor 638 also interacts with further device subsystems or resources such as the display 622, flash memory 624, random access memory (RAM) 626, auxiliary input/output (I/O) subsystems 628, serial port 630, keyboard 632, speaker 634, microphone 636, a short-range communications subsystem 640 and any other device subsystems generally designated as 642. APIs, including sensitive APIs requiring verification of one or more corresponding digital signatures before access is granted, may be provided on the device 610 to interface between software applications and any of the resources shown in Fig. 6.

Some of the subsystems shown in Fig. 6 perform communication-related functions, whereas other subsystems may provide "resident" or on-device functions. Notably, some subsystems, such as keyboard 632 and display 622 for example, may be used for both communication-related functions, such as entering a text message for transmission over a communication network, and device-resident functions such as a calculator or task list.

Operating system software used by the microprocessor 638, and possibly APIs to be accessed by software applications, is preferably stored in a persistent store such as flash memory 624, which may instead be a read only memory (ROM) or similar storage element (not shown). Those skilled in the art will appreciate that the operating system, specific device software applications, or parts thereof, may be temporarily loaded into a volatile store such as RAM 626. It is contemplated that received and transmitted communication signals may also be stored to RAM 626.

The microprocessor 638, in addition to its operating system functions, preferably enables execution of software applications on the device. A predetermined set of applications which control basic device operations, including at least data and voice communication applications for example, will normally be installed on the device 610 during manufacture. A preferred application that may be loaded onto the device may be a personal information manager (PIM) application having the ability to organize and manage data items relating to the device user such as, but not limited to e-mail, calendar events, voice mails, appointments, and task items. Naturally, one or more memory stores would be available on the device to facilitate storage of PIM data items on the device. Such PIM application would preferably have the ability to send and receive data items, via the wireless network. In a preferred embodiment, the PIM data items are seamlessly integrated, synchronized and updated, via the wireless network, with the device user's corresponding data items stored or associated with a host computer system thereby creating a mirrored host computer on the mobile device with respect to the data items at least. This would be especially advantageous in the case where the host computer system is the mobile device user's office computer system. Further applications, including signed software

applications as described above, may also be loaded onto the device 610 through the network 619, an auxiliary I/O subsystem 628, serial port 630, short-range communications subsystem 640 or any other suitable subsystem 642. The device microprocessor 638 may then verify any digital signatures, possibly including both "global" device signatures and API-specific signatures, appended to such a software application before the software application can be executed by the microprocessor 638 and/or access any associated sensitive APIs. Such flexibility in application installation increases the functionality of the device and may provide enhanced on-device functions, communication-related functions, or both. For example, secure communication applications may enable electronic commerce functions and other such financial transactions to be performed using the device 610, through a crypto API and a crypto module which implements crypto algorithms on the device (not shown).

In a data communication mode, a received signal such as a text message or web page download will be processed by the communication subsystem 611 and input to the microprocessor 638, which will preferably further process the received signal for output to the display 622, or alternatively to an auxiliary I/O device 628. A user of device 610 may also compose data items such as email messages for example, using the keyboard 632, which is preferably a complete alphanumeric keyboard or telephone-type keypad, in conjunction with the display 622 and possibly an auxiliary I/O device 628. Such composed items may then be transmitted over a communication network through the communication subsystem 611.

For voice communications, overall operation of the device 610 is substantially similar, except that received signals would preferably be output to a speaker 634 and signals for transmission would be generated by a microphone 636. Alternative voice or audio I/O

subsystems such as a voice message recording subsystem may also be implemented on the device 610. Although voice or audio signal output is preferably accomplished primarily through the speaker 634, the display 622 may also be used to provide an indication of the identity of a calling party, the duration of a voice call, or other voice call related information for example.

The serial port 630 in Fig. 6 would normally be implemented in a personal digital assistant (PDA)-type communication device for which synchronization with a user's desktop computer (not shown) may be desirable, but is an optional device component. Such a port 630 would enable a user to set preferences through an external device or software application and would extend the capabilities of the device by providing for information or software downloads to the device 610 other than through a wireless communication network. The alternate download path may for example be used to load an encryption key onto the device through a direct and thus reliable and trusted connection to thereby enable secure device communication.

A short-range communications subsystem 640 is a further optional component which may provide for communication between the device 624 and different systems or devices, which need not necessarily be similar devices. For example, the subsystem 640 may include an infrared device and associated circuits and components or a Bluetooth™ communication module to provide for communication with similarly-enabled systems and devices.

The embodiments described herein are examples of structures, systems or methods having elements corresponding to the elements of the invention recited in the claims. This written description may enable those skilled in the art to make and use embodiments having alternative elements that likewise correspond to the elements of the invention recited in the claims. The intended scope of the invention thus includes other structures, systems or methods

that do not differ from the literal language of the claims, and further includes other structures, systems or methods with insubstantial differences from the literal language of the claims.

For example, when a software application is rejected at step 250 in the method shown in Fig. 5, the signing authority may request that the developer sign a contract or enter into a business relationship with a device manufacturer or other entity on whose behalf the signing authority acts. Similarly, if a software application is rejected at step 270, authority to sign the software application may be delegated to a different signing authority. The signing of a software application following delegation of signing of the software application to the different authority can proceed substantially as shown in Fig. 5, wherein the target signing authority that received the original request from the trusted developer at step 220 requests that the software application be signed by the different signing authority on behalf of the trusted developer from the target signing authority. Once a trust relationship has been established between code signing authorities, target private code signing keys could be shared between code signing authorities to improve performance of the method at step 240, or a device may be configured to validate digital signatures from either of the trusted signing authorities.

In addition, although described primarily in the context of software applications, code signing systems and methods according to the present invention may also be applied to other device-related components, including but in no way limited to, commands and associated command arguments, and libraries configured to interface with device resources. Such commands and libraries may be sent to mobile devices by device manufacturers, device owners, network operators, service providers, software application developers and the like. It would be desirable to control the execution of any command that may affect device operation, such as a

command to change a device identification code or wireless communication network address for example, by requiring verification of one or more digital signatures before a command can be executed on a device, in accordance with the code signing systems and methods described and claimed herein.

We claim:

1. A code signing system for operation in conjunction with a software application having a digital signature, comprising:

an application platform;

an application programming interface (API) configured to link the software application with the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application.

2. The code signing system of claim 1, wherein the virtual machine denies the software application access to the API if the digital signature is not authentic.

3. The code signing system of claim 1, wherein the virtual machine purges the software application if the digital signature is not authentic.

4. The code signing system of claim 1, wherein the code signing system is installed on a mobile device.

5. The code signing system of claim 1, wherein the digital signature is generated by a code signing authority.

6. A code signing system for operation in conjunction with a software application having a digital signature, comprising:

an application platform;

a plurality of application programming interfaces (APIs), each configured to link the software application with a resource on the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application,

wherein the virtual machine verifies the authenticity of the digital signature in order to control access to the plurality of APIs by the software application.

7. The code signing system of claim 6, wherein the plurality of APIs are included in an API library.

8. The code signing system of claim 6, wherein one or more of the plurality of APIs is classified as sensitive, and wherein the virtual machine uses the digital signature to control access to the sensitive APIs.

9. The code signing system of claim 8, for operation in conjunction with a plurality of software applications, wherein one or more of the plurality of software applications has a digital signature, and wherein the virtual machine verifies the authenticity of the digital signature of each of the one or more of the plurality of software applications in order to control access to the sensitive APIs by each of the plurality of software applications.

10. The code signing system of claim 6, wherein the resource on the application platform comprises a wireless communication system.

11. The code signing system of claim 6, wherein the resource on the application platform comprises a cryptographic module which implements cryptographic algorithms.

12. The code signing system of claim 6, wherein the resource on the application platform comprises a data store.

13. The code signing system of claim 6, wherein the resource on the application platform comprises a user interface (UI).

14. The code signing system of claim 1, further comprising:

a plurality of API libraries each including a plurality of APIs, wherein the virtual machine controls access to the plurality of API libraries by the software application.

15. The code signing system of claim 14, wherein one or more of the plurality of API libraries is classified as sensitive, and wherein the virtual machine uses the digital signature to control access to the sensitive API libraries by the software application.

16. The code signing system of claim 15, wherein the software application includes a unique digital signature for each sensitive API library.

17. The code signing system of claim 16, wherein:

the software application includes a signature identification for each unique digital signature;

each sensitive API library includes a signature identifier; and

the virtual machine compares the signature identification and the signature identifier to match the unique digital signatures with sensitive API libraries.

18. The code signing system of claim 1, wherein the digital signature is generated using a private signature key, and the virtual machine uses a public signature key to verify the authenticity of the digital signature.

19. The code signing system of claim 18, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application; and

the virtual machine verifies the authenticity of the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and comparing the generated hash with the recovered hash.

20. The code signing system of claim 1, wherein the API further comprises:

a description string that is displayed by the mobile device when the software application attempts to access the API.

21. The code signing system of claim 1, wherein the application platform comprises an operating system.

22. The code signing system of claim 1, wherein the application platform comprises one or more core functions of a mobile device.

23. The code signing system of claim 1, wherein the application platform comprises hardware on a mobile device.

24. The code signing system of claim 23, wherein the hardware comprises a subscriber identity module (SIM) card.

25. The code signing system of claim 1, wherein the software application is a Java application for a mobile device.

26. The code signing system of claim 1, wherein the API interfaces with a cryptographic routine on the application platform.

27. The code signing system of claim 1, wherein the API interfaces with a proprietary data model on the application platform.

28. The code signing system of claim 1, wherein the virtual machine is a Java virtual machine installed on a mobile device.

29. A method of controlling access to sensitive application programming interfaces on a mobile device, comprising the steps of:

loading a software application on the mobile device that requires access to a sensitive application programming interface (API);

determining whether or not the software application includes a digital signature associated with the sensitive API; and

if the software application does not include a digital signature associated with the sensitive API, then denying the software application access to the sensitive API.

30. The method of claim 29, comprising the additional step of:

if the software application does not include a digital signature associated with the sensitive API, then purging the software application from the mobile device.

31. The method of claim 29, wherein the digital signature is generated by a code signing authority.

32. The method of claim 29, comprising the additional steps of:

if the software application includes a digital signature associated with the sensitive API, then verifying the authenticity of the digital signature; and

if the digital signature is not authentic, then denying the software application access to the sensitive API.

33. The method of claim 32, comprising the additional step of:

if the digital signature is not authentic, then purging the software application from the mobile device.

34. The method of claim 32, wherein the digital signature is generated by applying a private signature key to a hash of the software application, and wherein the step of verifying the authenticity of the digital signature is performed by a method comprising the steps of:

storing a public signature key that corresponds to the private signature key on the mobile device;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

35. The method of claim 34, wherein the digital signature is generated by calculating a hash of the software application and applying the private signature key.

36. The method of claim 29, comprising the additional step of:

displaying a description string that notifies a user of the mobile device that the software application requires access to the sensitive API.

37. The method of claim 36, comprising the additional step of:

receiving a command from the user granting or denying the software application access to the sensitive API.

38. A method of controlling access to an application programming interface (API) on a mobile device by a software application created by a software developer, comprising the steps of:

receiving the software application from the software developer;

reviewing the software application to determine if it may access the API;

if the software application may access the API, then appending a digital signature to the software application;

verifying the authenticity of a digital signature appended to a software application; and

providing access to the API to software applications for which the appended digital signature is authentic.

39. The method of claim 38, wherein the step of reviewing the software application is performed by a code signing authority.

40. The method of claim 38, wherein the step of appending the digital signature to the software application is performed by a method comprising the steps of:

calculating a hash of the software application; and

applying a signature key to the hash of the software application to generate the digital signature.

41. The method of claim 40, wherein the hash of the software application is calculated using the Secure Hash Algorithm (SHA1).

42. The method of claim 40, wherein the step of verifying the authenticity of a digital signature comprises the steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash; and

determining if the digital signature is authentic by comparing the calculated hash with the recovered hash.

43. The method of claim 42, comprising the further step of, if the digital signature is not authentic, then denying the software application access to the API.

44. The method of claim 42, wherein the signature key is a private signature key and the corresponding signature key is a public signature key.

45. A method of controlling access to a sensitive application programming interface (API) on a mobile device, comprising the steps of:

registering one or more software developers that are trusted to design software applications which access the sensitive API;

receiving a hash of a software application;

determining if the software application was designed by one of the registered software developers; and

if the software application was designed by one of the registered software developers, then generating a digital signature using the hash of the software application,

wherein

the digital signature may be appended to the software application; and

the mobile device verifies the authenticity of the digital signature in order to control access to the sensitive API by the software application.

46. The method of claim 45, wherein the step of generating the digital signature is performed by a code signing authority.

47. The method of claim 45, wherein the step of generating the digital signature is performed by applying a signature key to the hash of the software application.

48. The method of claim 47, wherein the mobile device verifies the authenticity of the digital signature by performing the additional steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash;

determining if the digital signature is authentic by comparing the calculated hash with the recovered hash; and

if the digital signature is not authentic, then denying the software application access to the sensitive API.

49. A method of restricting access to application programming interfaces on a mobile device, comprising the steps of:

loading a software application on the mobile device that requires access to one or more application programming interface (API);

determining whether or not the software application includes an authentic digital signature associated with the mobile device; and

if the software application does not include an authentic digital signature associated with the mobile device, then denying the software application access to the one or more APIs.

50. The method of claim 49, comprising the additional step of:

if the software application does not include an authentic digital signature associated with the mobile device, then purging the software application from the mobile device.

51. The method of claim 49, wherein:

the software application includes a plurality of digital signatures; and

the plurality of digital signatures includes digital signatures respectively associated with different types of mobile devices.

52. The method of claim 51, wherein each of the plurality of digital signatures is generated by a respective corresponding code signing authority.

53. The method of claim 49, wherein the step of determining whether or not the software application includes an authentic digital signature associated with the mobile device comprises the additional steps of:

determining if the software application includes a digital signature associated with the mobile device; and

if so, then verifying the authenticity of the digital signature.

54. The method of claim 53, wherein the one or more APIs includes one or more APIs classified as sensitive, and the method further comprises the steps of, for each sensitive API:

determining whether or not the software application includes an authentic digital signature associated with the sensitive API; and

if the software application does not include an authentic digital signature associated with the sensitive API, then denying the software application access to the sensitive API.

55. The method of claim 52, wherein each of the plurality of digital signatures is generated by its corresponding code signing authority by applying a respective private signature key associated with the code signing authority to a hash of the software application.

56. The method of claim 55, wherein the step of determining whether or not the software application includes an authentic digital signature associated with the mobile device comprises the steps of:

determining if the software application includes a digital signature associated with the mobile device; and

if so, then verifying the authenticity of the digital signature,

wherein the step of verifying the authenticity of the digital signature is performed by a method comprising the steps of:

storing a public signature key on a mobile device that corresponds to the private signature key associated with the code signing authority which generates the signature associated with the mobile device;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

ABSTRACT

A code signing system and method is provided. The code signing system operates in conjunction with a signed software application having a digital signature and includes an application platform, an application programming interface (API), and a virtual machine. The API is configured to link the software application with the application platform. The virtual machine verifies the authenticity of the digital signature in order to control access to the API by the software application.

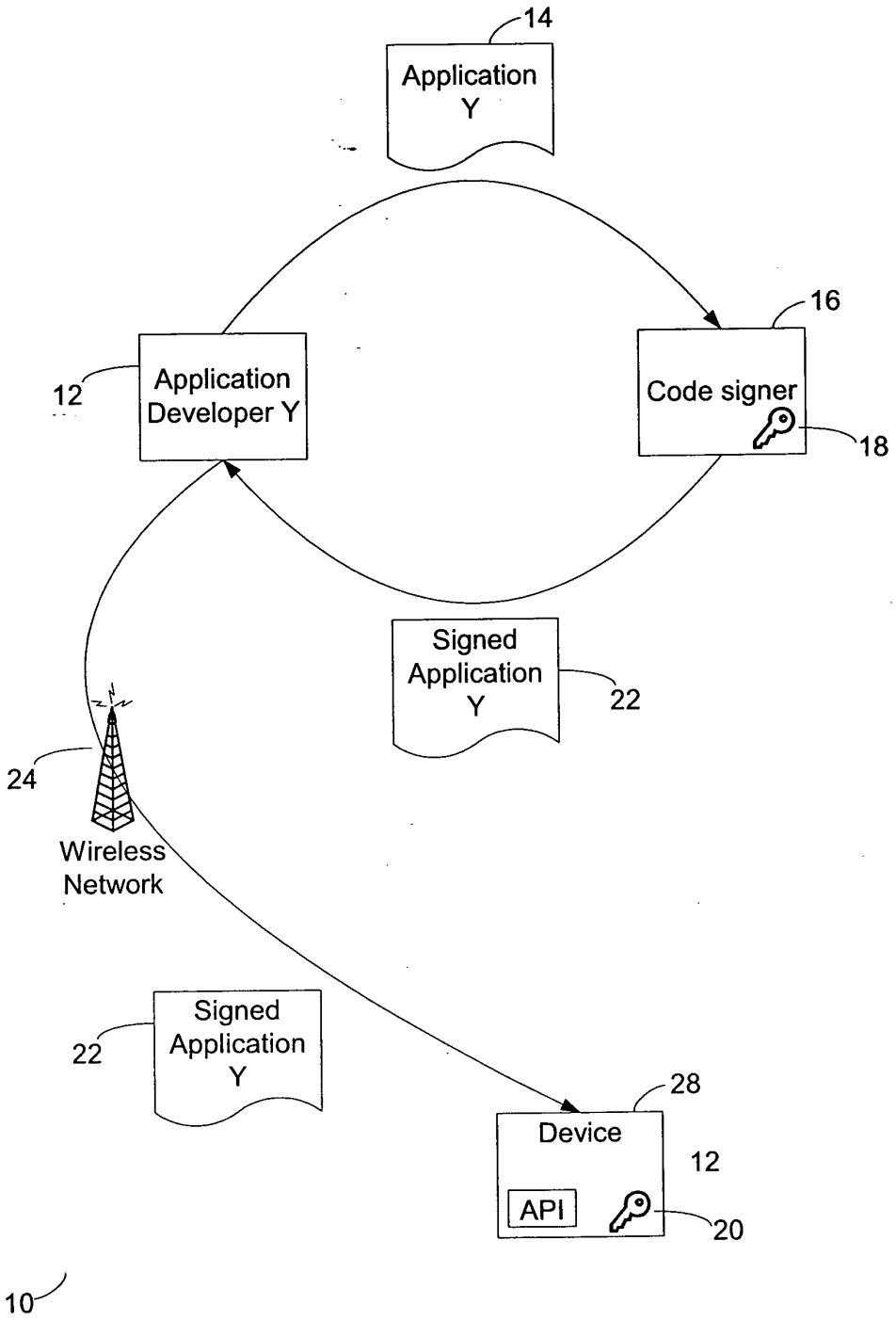
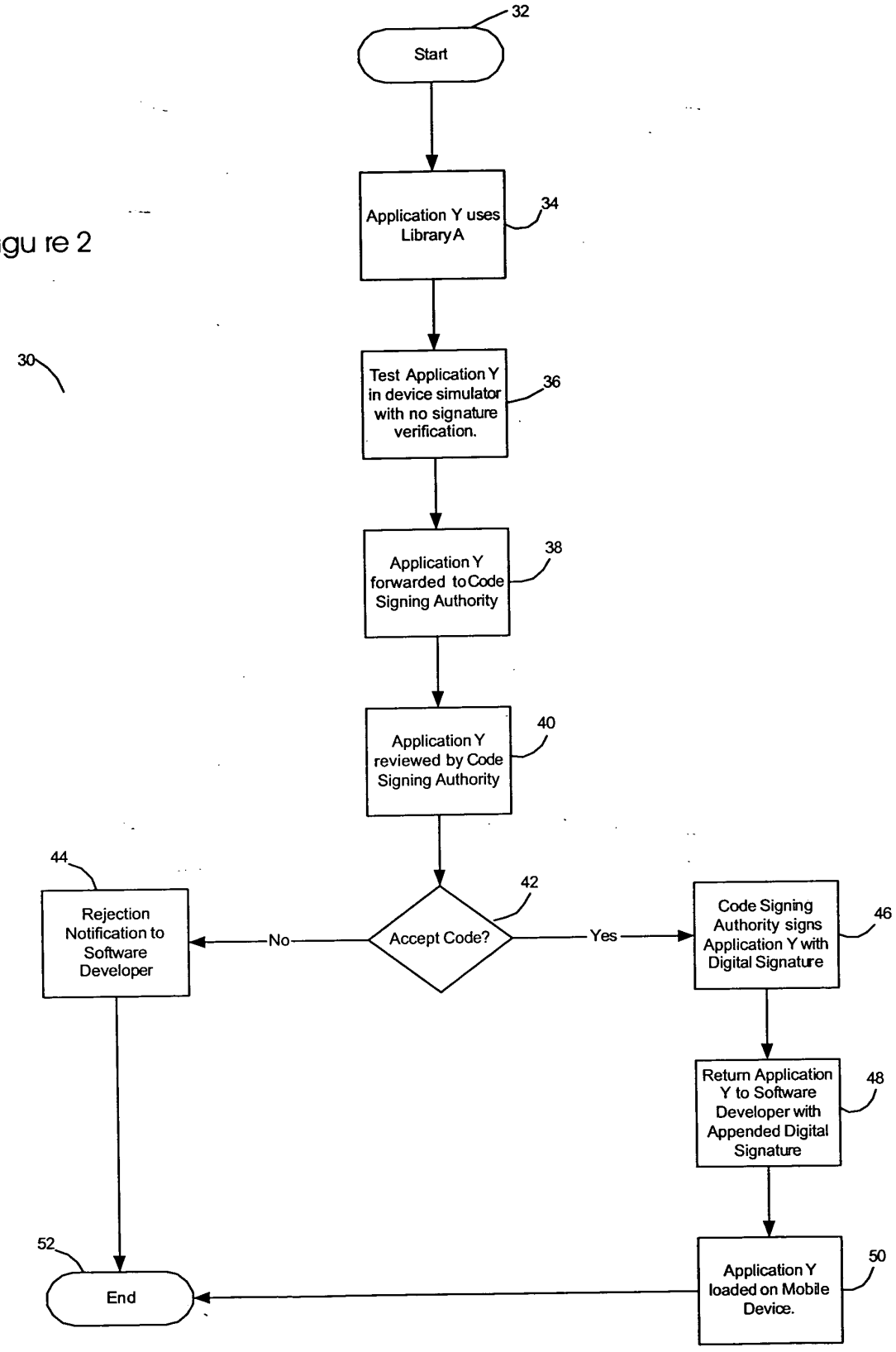
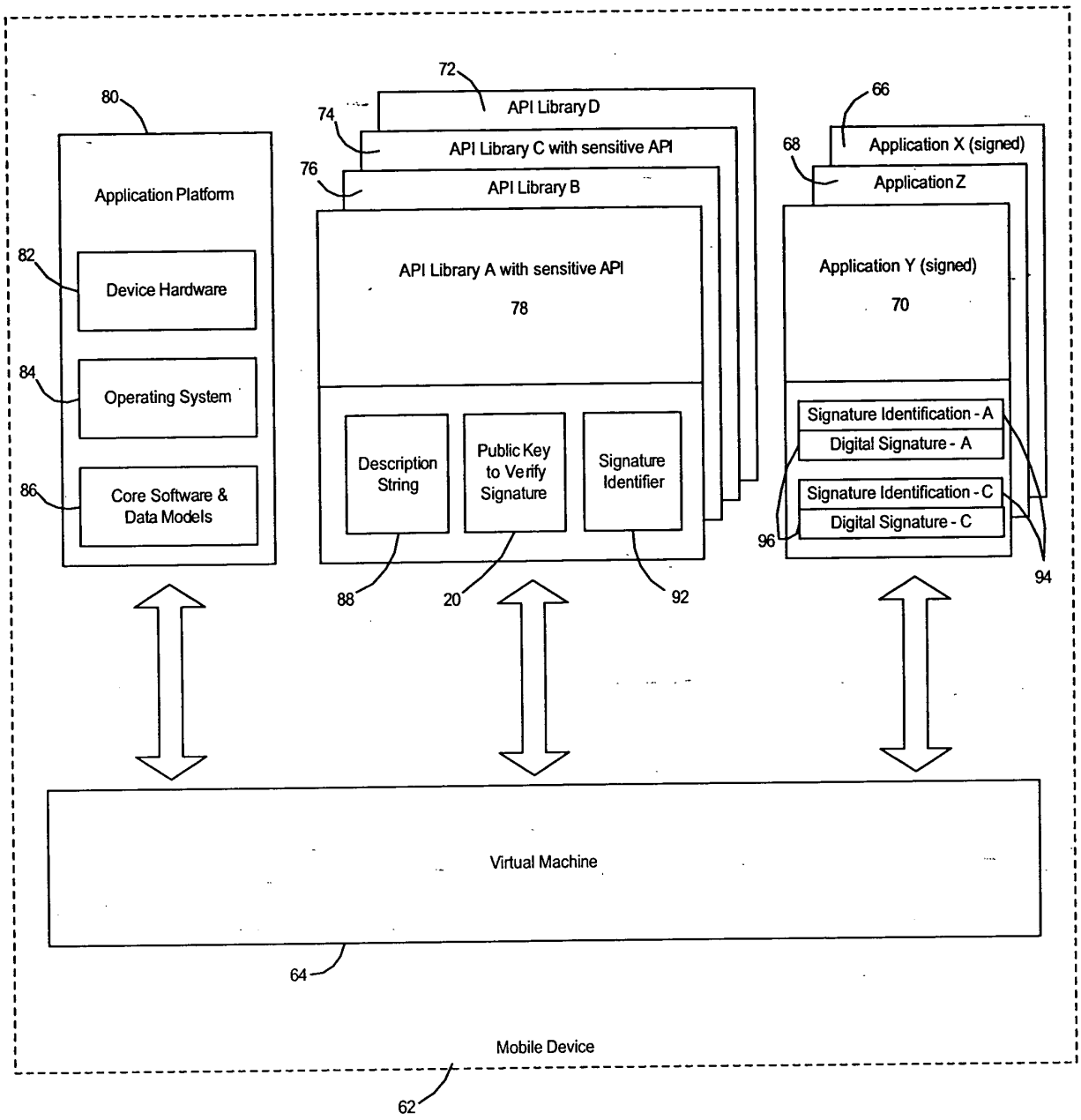

14
Application Y
16
Application Developer Y
12
Code signer
18
Signed Application Y
22
24
Wireless Network
Signed Application Y
22
28
Device
12
API
20
10

Figure 1

Figu re 2


30
Start
32
Application Y uses Library A
34
Test Application Y in device simulator with no signature verification.
36
Application Y forwarded to Code Signing Authority
38
Application Y reviewed by Code Signing Authority
40
Accept Code?
42
No
Yes
Rejection Notification to Software Developer
44
Code Signing Authority signs Application Y with Digital Signature
46
Return Application Y to Software Developer with Appended Digital Signature
48
Application Y loaded on Mobile Device.
50
End
52


80
Application Platform
82
Device Hardware
84
Operating System
86
Core Software & Data Models
72
API Library D
74
API Library C with sensitive API
76
API Library B
API Library A with sensitive API
78
Description String
Public Key to Verify Signature
Signature Identifier
88
20
92
66
68
Application X (signed)
Application Z
Application Y (signed)
70
Signature Identification - A
Digital Signature - A
Signature Identification - C
Digital Signature - C
96
94
Virtual Machine
64
Mobile Device
62
60

Figu re 3

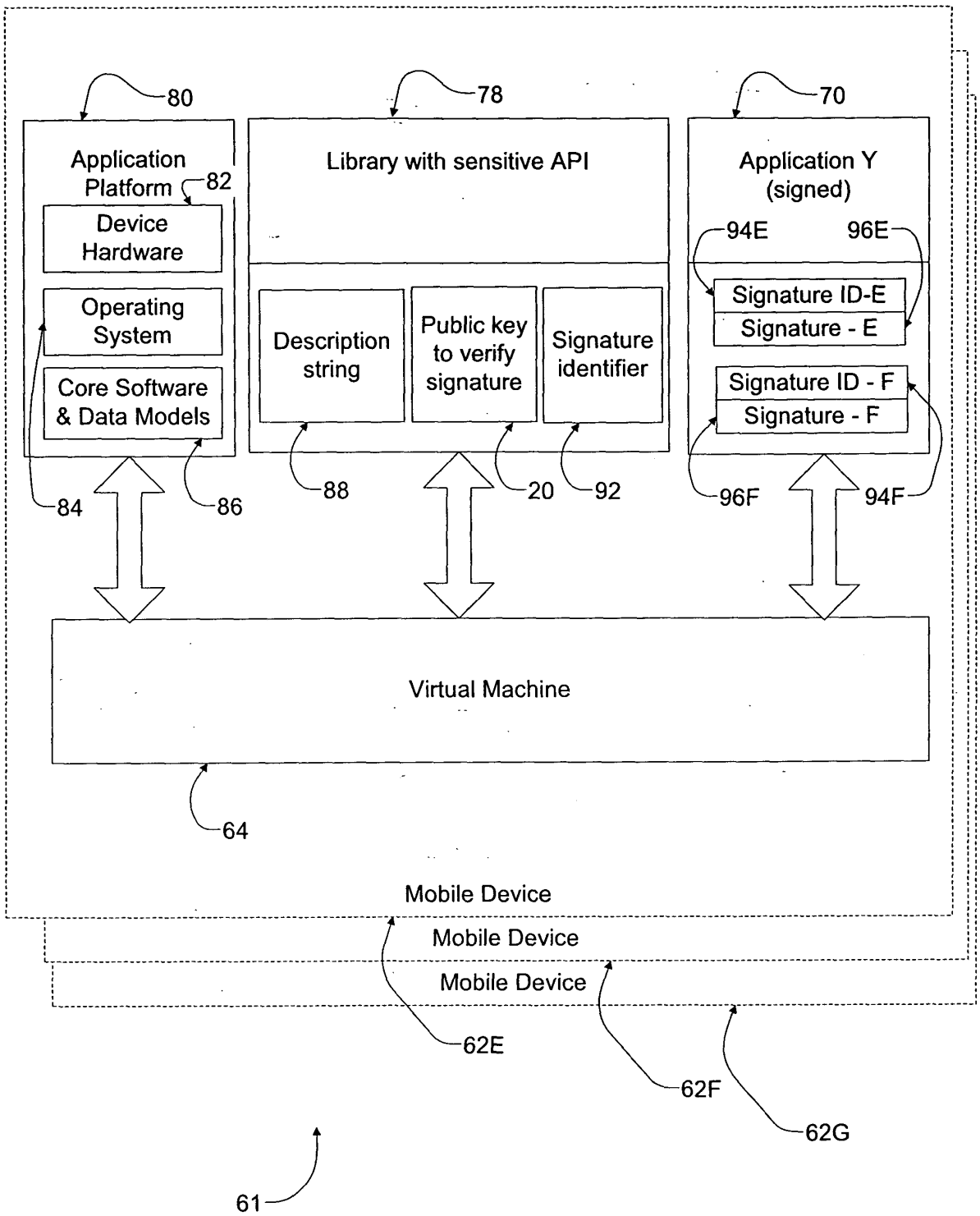
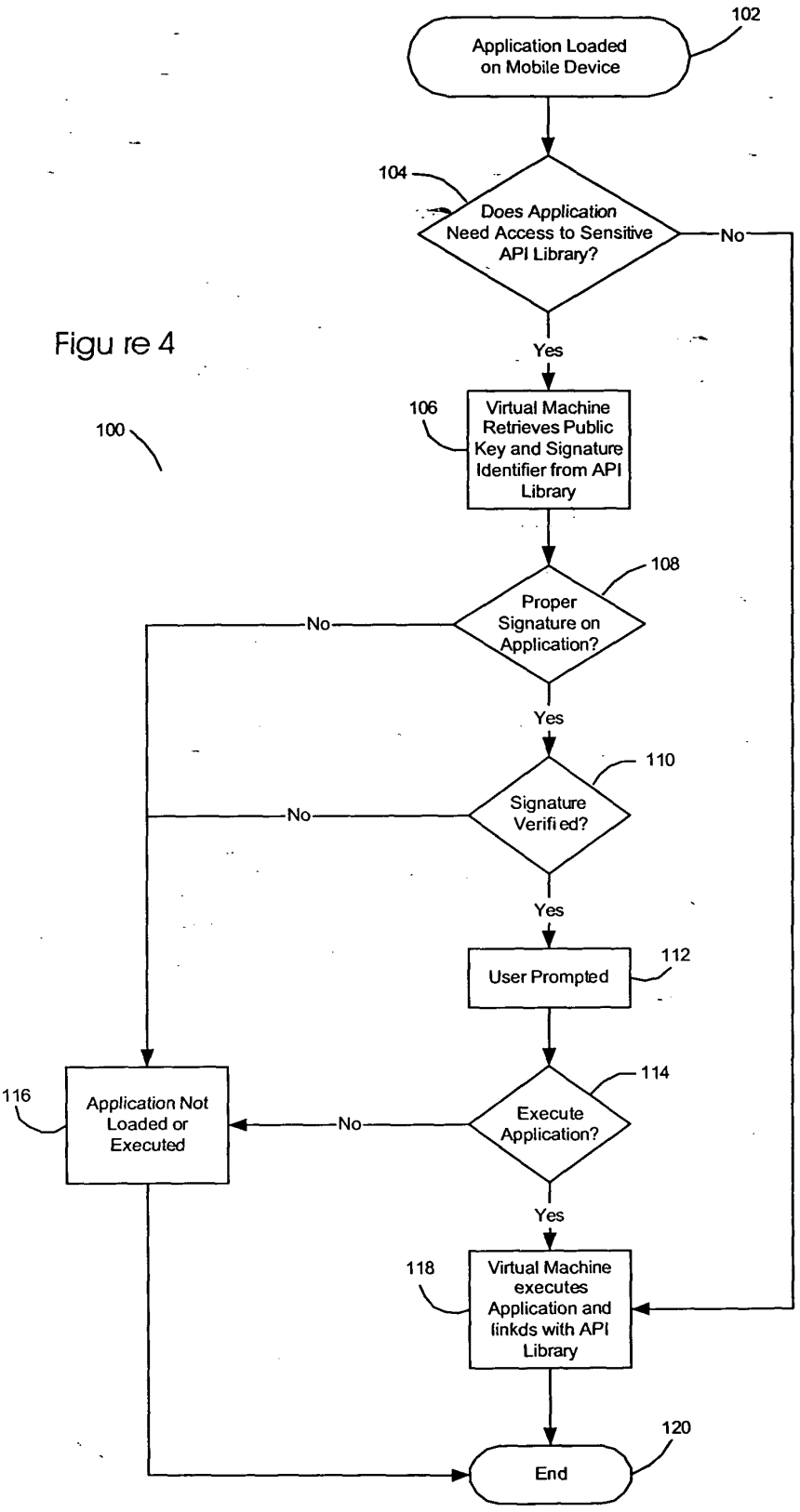

80
78
70
Application Platform
82
Device Hardware
Operating System
Core Software & Data Models
Library with sensitive API
Description string
Public key to verify signature
Signature identifier
Application Y (signed)
94E
96E
Signature ID-E
Signature - E
Signature ID - F
Signature - F
84
86
88
20
92
96F
94F
Virtual Machine
64
Mobile Device
Mobile Device
Mobile Device
62E
62F
62G
61

Figure 3A

Figu re 4


100
102
Application Loaded on Mobile Device
104
Does Application Need Access to Sensitive API Library?
No
Yes
106
Virtual Machine Retrieves Public Key and Signature Identifier from API Library
108
Proper Signature on Application?
No
Yes
110
Signature Verified?
No
Yes
112
User Prompted
114
Execute Application?
No
116
Application Not Loaded or Executed
Yes
118
Virtual Machine executes Application and linkds with API Library
120
End

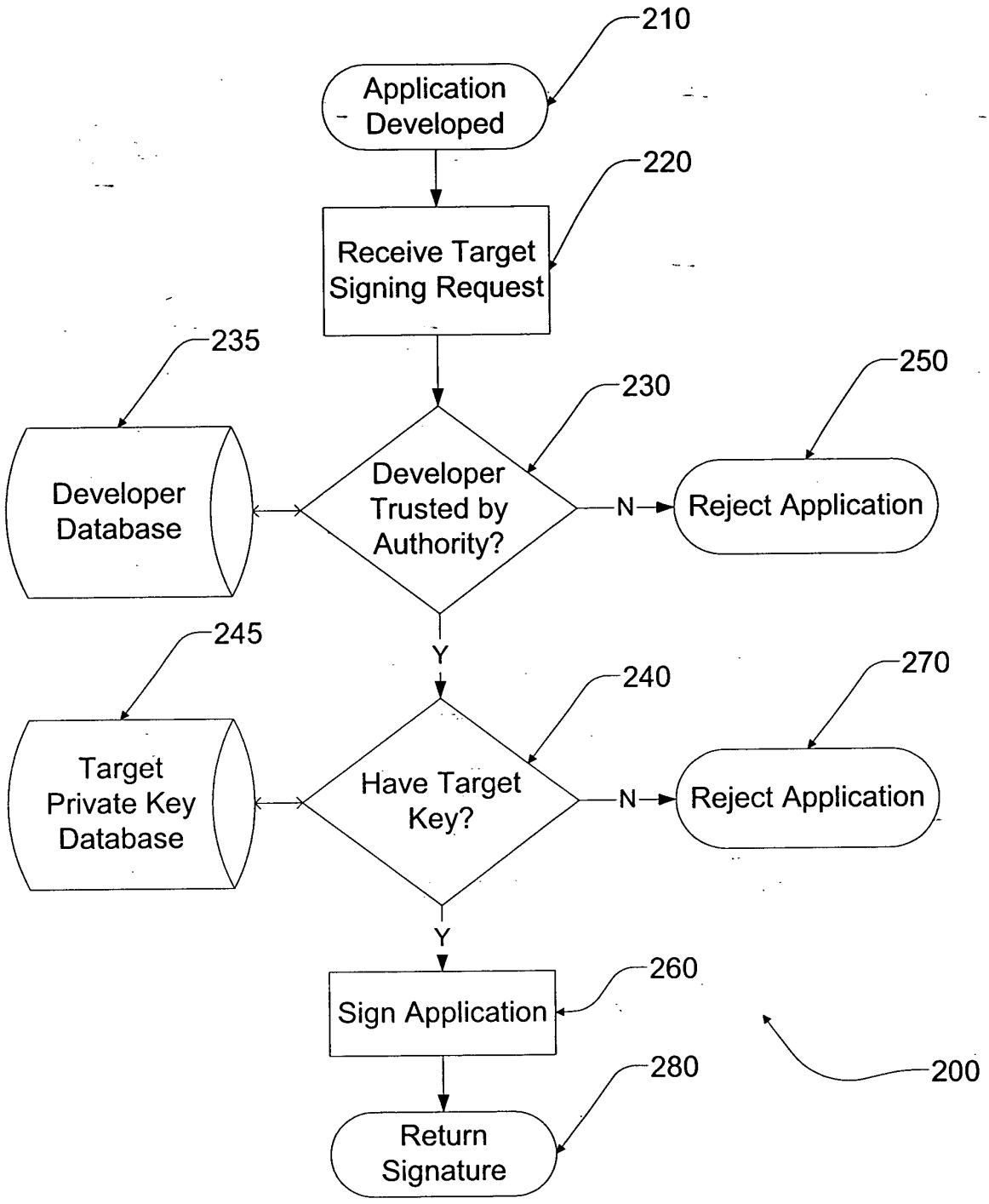

210
Application Developed
220
Receive Target Signing Request
235
Developer Database
230
Developer Trusted by Authority?
N
250
Reject Application
Y
245
Target Private Key Database
240
Have Target Key?
N
270
Reject Application
Y
260
Sign Application
280
Return Signature
200

Figure 5

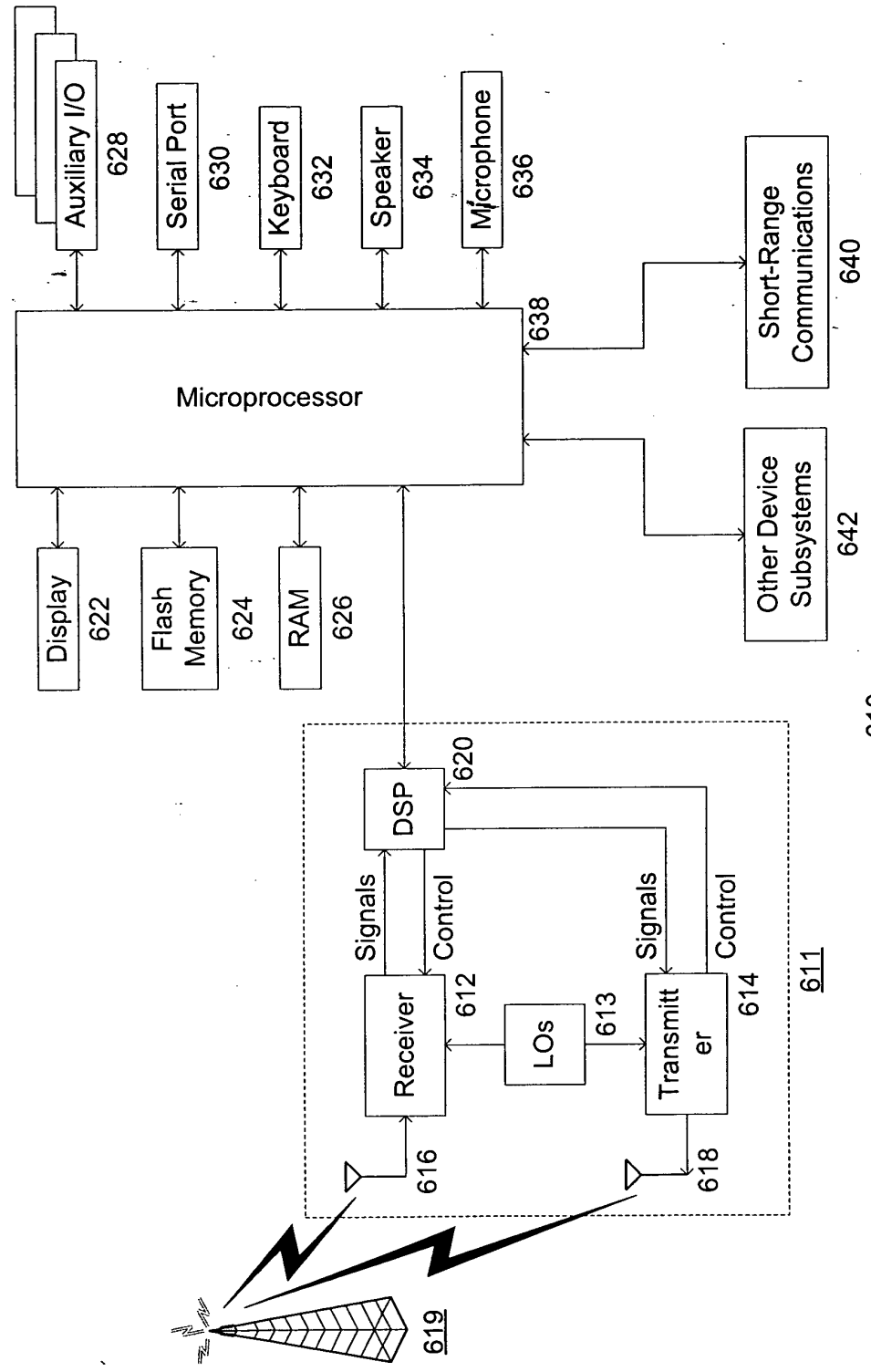

Auxiliary I/O
628
Serial Port
630
Keyboard
632
Speaker
634
Microphone
636
Microprocessor
638
Short-Range Communications
640
Other Device Subsystems
642
Display
622
Flash Memory
624
RAM
626
610
DSP
620
Signals
Control
Signals
Control
Receiver
612
LOs
613
Transmitter
614
611
616
618
619

Figure 6

PATENT COOPERATION TREATY

PCT

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

(PCT Article 36 and Rule 70)

Applicant's or agent's file reference	FOR FURTHER ACTION	See Notification of Transmittal of International Preliminary Examination Report (Form PCT/IPEA/416)
PWO-0445		
International application No.	International filing date *(day/month/year)*	Priority date *(day/month/year)*
PCT/CA01/01344	20/09/2001	21/09/2000
International Patent Classification (IPC) or national classification and IPC G06F1/00		
Applicant RESEARCH IN MOTION LIMITED et al.		

1. This international preliminary examination report has been prepared by this International Preliminary Examining Authority and is transmitted to the applicant according to Article 36.

2. This REPORT consists of a total of 4 sheets, including this cover sheet.

   ☐ This report is also accompanied by ANNEXES, i.e. sheets of the description, claims and/or drawings which have been amended and are the basis for this report and/or sheets containing rectifications made before this Authority (see Rule 70.16 and Section 607 of the Administrative Instructions under the PCT).

   These annexes consist of a total of sheets.

3. This report contains indications relating to the following items:

   - I ☒ Basis of the report
   - II ☐ Priority
   - III ☒ Non-establishment of opinion with regard to novelty, inventive step and industrial applicability
   - IV ☐ Lack of unity of invention
   - V ☐ Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations suporting such statement
   - VI ☐ Certain documents cited
   - VII ☐ Certain defects in the international application
   - VIII ☐ Certain observations on the international application

Date of submission of the demand	Date of completion of this report
18/04/2002	15.11.2002
Name and mailing address of the international preliminary examining authority: European Patent Office D-80298 Munich Tel. +49 89 2399 - 0 Tx: 523656 epmu d Fax: +49 89 2399 - 4465	Authorized officer Kerschbaumer, J Telephone No. +49 89 2399 2999

Form PCT/IPEA/409 (cover sheet) (January 1994)

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No. PCT/CA01/01344

I. **Basis of the report**

1. With regard to the **elements** of the international application *(Replacement sheets which have been furnished to the receiving Office in response to an invitation under Article 14 are referred to in this report as "originally filed" and are not annexed to this report since they do not contain amendments (Rules 70.16 and 70.17))*:
Description, pages:

1-28	as originally filed				

Claims, No.:

1-109	as received on	28/06/2002	with letter of	28/06/2002	

Drawings, sheets:

1/7-7/7	as originally filed

2. With regard to the **language**, all the elements marked above were available or furnished to this Authority in the language in which the international application was filed, unless otherwise indicated under this item.

   These elements were available or furnished to this Authority in the following language: , which is:

   ☐ the language of a translation furnished for the purposes of the international search (under Rule 23.1(b)).
   ☐ the language of publication of the international application (under Rule 48.3(b)).
   ☐ the language of a translation furnished for the purposes of international preliminary examination (under Rule 55.2 and/or 55.3).

3. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, the international preliminary examination was carried out on the basis of the sequence listing:

   ☐ contained in the international application in written form.
   ☐ filed together with the international application in computer readable form.
   ☐ furnished subsequently to this Authority in written form.
   ☐ furnished subsequently to this Authority in computer readable form.
   ☐ The statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.
   ☐ The statement that the information recorded in computer readable form is identical to the written sequence listing has been furnished.

4. The amendments have resulted in the cancellation of:

   ☐ the description, pages:
   ☐ the claims, Nos.:

☐ the drawings, sheets:

5. ☐ This report has been established as if (some of) the amendments had not been made, since they have been considered to go beyond the disclosure as filed (Rule 70.2(c)):

(Any replacement sheet containing such amendments must be referred to under item 1 and annexed to this report.)

6. Additional observations, if necessary:

III. Non-establishment of opinion with regard to novelty, inventive step and industrial applicability

1. The questions whether the claimed invention appears to be novel, to involve an inventive step (to be non-obvious), or to be industrially applicable have not been examined in respect of:

☒ the entire international application.

☐ claims Nos. .

because:

☐ the said international application, or the said claims Nos. relate to the following subject matter which does not require an international preliminary examination (*specify*):

☒ the description, claims or drawings (*indicate particular elements below*) or said claims Nos. are so unclear that no meaningful opinion could be formed (*specify*):
see separate sheet

☐ the claims, or said claims Nos. are so inadequately supported by the description that no meaningful opinion could be formed.

☐ no international search report has been established for the said claims Nos. .

2. A meaningful international preliminary examination cannot be carried out due to the failure of the nucleotide and/or amino acid sequence listing to comply with the standard provided for in Annex C of the Administrative Instructions:

☐ the written form has not been furnished or does not comply with the standard.

☐ the computer readable form has not been furnished or does not comply with the standard.

Re Item III

Although system claims 1, 6, 56, 77 and method claims 27, 36, 43, 47, 68, 87, 104 have been drafted as separate independent claims, they appear to relate effectively to the same subject-matter and to differ from each other only with regard to the definition of the subject-matter for which protection is sought or in respect of the terminology used for the features of that subject-matter. The aforementioned claims therefore lack conciseness. Moreover, lack of clarity of the claims as a whole arises, since the plurality of independent claims makes it impossible to determine the matter for which protection is sought, and places an undue burden on others seeking to establish the extent of the protection.

Hence, system claims 1, 6, 56, 77 and method claims 27, 36, 43, 47, 68, 87, 104 do not meet the requirements of Article 6 PCT.

We claim:

1. A code signing system for operation in conjunction with a software application having a digital signature and a signature identification, where the digital signature is associated with the signature identification, comprising:

an application platform;

an application programming interface (API) having an associated signature identifier, the API is configured to link the software application with the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the API by the software application where the signature identifier corresponds to the signature identification.

2. The code signing system of claim 1, wherein the virtual machine denies the software application access to the API if the digital signature is not authenticated.

3. The code signing system of claim 1, wherein the virtual machine purges the software application if the digital signature is not authenticated.

4. The code signing system of claim 1, wherein the code signing system is installed on a mobile device.

5. The code signing system of claim 1, wherein the digital signature is generated by a code signing authority.

6. A code signing system for operation in conjunction with a software application having a digital signature and a signature identification where the digital signature is associated with the signature identification, comprising:

an application platform;

a plurality of application programming interfaces (APIs) associated with a signature identifier, each configured to link the software application with a resource on the application platform; and

a virtual machine that verifies the authenticity of the digital signature in order to control access to the APIs by the software application where the signature identification corresponds to the signature identifier,

wherein the virtual machine verifies the authenticity of the digital signature in order to control access to the plurality of APIs by the software application.

7. The code signing system of claim 6, wherein the plurality of APIs are included in an API library.

8. The code signing system of claim 6, wherein one or more of the plurality of APIs is classified as sensitive and having an associated signature identifier, and wherein the virtual machine uses the digital signature and the signature identification to control access to the sensitive APIs.

9. The code signing system of claim 8, wherein the code signing system operates in conjunction with a plurality of software applications, wherein one or more of the plurality of software applications has a digital signature and a signature identification, and wherein the virtual machine verifies the authenticity of the digital signature of each of the one or more of the plurality of software applications, where the signature identification corresponds to the signature identifier of the respective sensitive APIs, in order to control access to the sensitive APIs by each of the plurality of software applications.

10. The code signing system of claim 6, wherein the resource on the application platform comprises a wireless communication system.

11. The code signing system of claim 6, wherein the resource on the application platform comprises a cryptographic module which implements cryptographic algorithms.

12. The code signing system of claim 6, wherein the resource on the application platform comprises a data store.

AMENDED SHEET

13. The code signing system of claim 6, wherein the resource on the application platform comprises a user interface (UI).

14. The code signing system of claim 1, further comprising:

a plurality of API libraries, each of the plurality of API libraries includes a plurality of APIs, wherein the virtual machine controls access to the plurality of API libraries by the software application.

15. The code signing system of claim 14, wherein at least one of the plurality of API libraries is classified as sensitive,

wherein access to a sensitive API library requires a digital signature associated with a signature identification where the signature identification corresponds to a signature identifier associated with the sensitive API library;

wherein the software application includes at least one digital signature and at least one associated signature identification for accessing sensitive API libraries; and

wherein the virtual machine authenticates the software application for accessing the sensitive API library by verifying the one digital signature included in the software application that has a signature identification corresponding to the signature identifier of the sensitive API library.

16. The code signing system of claim 1, wherein the digital signature is generated using a private signature key, and the virtual machine uses a public signature key to verify the authenticity of the digital signature.

17. The code signing system of claim 16, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application; and

the virtual machine verifies the authenticity of the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the

AMENDED SHEET

digital signature to obtain a recovered hash, and comparing the generated hash with the recovered hash.

18. The code signing system of claim 4, wherein the API further comprises:

a description string that is displayed by the mobile device when the software application attempts to access the API.

19. The code signing system of claim 1, wherein the application platform comprises an operating system.

20. The code signing system of claim 1, wherein the application platform comprises one or more core functions of a mobile device.

21. The code signing system of claim 1, wherein the application platform comprises hardware on a mobile device.

22. The code signing system of claim 23, wherein the hardware comprises a subscriber identity module (SIM) card.

23. The code signing system of claim 1, wherein the software application is a Java application for a mobile device.

24. The code signing system of claim 1, wherein the API interfaces with a cryptographic routine on the application platform.

25. The code signing system of claim 1, wherein the API interfaces with a proprietary data model on the application platform.

26. The code signing system of claim 1, wherein the virtual machine is a Java virtual machine installed on a mobile device.

AMENDED SHEET

27. A method of controlling access to sensitive application programming interfaces on a mobile device, comprising the steps of:

loading a software application on the mobile device that requires access to a sensitive application programming interface (API) having a signature identifier;

determining whether the software application includes a digital signature and a signature identification; and

denying the software application access to the sensitive API where the signature identification does not correspond with the signature identifier. .

28. The method of claim 27, comprising the additional step of:

purging the software application from the mobile device where the signature identification does not correspond with the signature identifier..

29. The method of claim 27, wherein the digital signature and the signature identification are generated by a code signing authority.

30. The method of claim 27, comprising the additional steps of:

verifying the authenticity of the digital signature where the signature identification corresponds with the signature identifier.; and

denying the software application access to the sensitive API where the digital signature is not authenticated.

31. The method of claim 30, comprising the additional step of:

purging the software application from the mobile device where the digital signature is not authenticated..

32. The method of claim 30, wherein the digital signature is generated by applying a private signature key to a hash of the software application, and wherein the step of verifying the authenticity of the digital signature is performed by a method comprising the steps of:

AMENDED SHEET

storing a public signature key that corresponds to the private signature key on the mobile device;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

33. The method of claim 32, wherein the digital signature is generated by calculating a hash of the software application and applying the private signature key.

34. The method of claim 27, comprising the additional step of:

displaying a description string that notifies a user of the mobile device that the software application requires access to the sensitive API.

35. The method of claim 34, comprising the additional step of:

receiving a command from the user granting or denying the software application access to the sensitive API.

36. A method of controlling access to an application programming interface (API) having a signature identifier on a mobile device by a software application created by a software developer, comprising the steps of:

receiving the software application from the software developer;

determining whether the software application satisfies at least one criterion;

appending a digital signature and a signature identification to the software application where the software application satisfies at least one criterion;;

verifying the authenticity of the digital signature appended to the software application where the signature identification corresponds with the signature identifier; and

providing access to the API to software applications where the digital signature is authenticated.

AMENDED SHEET

37. The method of claim 36, wherein the step of determining whether the software application satisfies at least one criterion is performed by a code signing authority.

38. The method of claim 36, wherein the step of appending the digital signature and the signature identification to the software application includes generating the digital signature comprising the steps of:

calculating a hash of the software application; and

applying a signature key to the hash of the software application to generate the digital signature.

39. The method of claim 38, wherein the hash of the software application is calculated using the Secure Hash Algorithm (SHA1).

40. The method of claim 38, wherein the step of verifying the authenticity of the digital signature comprises the steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash; and

authenticating the digital signature by comparing the calculated hash with the recovered hash.

41. The method of claim 40, comprising the further step of denying the software application access to the API where the digital signature is not authenticated..

42. The method of claim 40, wherein the signature key is a private signature key and the corresponding signature key is a public signature key.

AMENDED SHEET

43. A method of controlling access to a sensitive application programming interface (API) having a signature identifier on a mobile device, comprising the steps of:

registering one or more software developers that are trusted to develop software applications which access the sensitive API;

receiving a hash of a software application;

determining whether the hash was sent by a registered software developer; and

generating a digital signature using the hash of the software application and a signature identification corresponding to the signature identifier where the hash was sent by the registered software developer;,

wherein

the digital signature and the signature identification are appended to the software application; and

the mobile device verifies the authenticity of the digital signature in order to control access to the sensitive API by the software application where the signature identification corresponds with the signature identifier.

44. The method of claim 43, wherein the step of generating the digital signature is performed by a code signing authority.

45. The method of claim 43, wherein the step of generating the digital signature is performed by applying a signature key to the hash of the software application.

46. The method of claim 45, wherein the mobile device verifies the authenticity of the digital signature by performing the additional steps of:

providing a corresponding signature key on the mobile device;

calculating the hash of the software application on the mobile device to obtain a calculated hash;

applying the corresponding signature key to the digital signature to obtain a recovered hash;

AMENDED SHEET

determining whether the digital signature is authentic by comparing the calculated hash with the recovered hash; and

denying the software application access to the sensitive API where the digital signature is not authenticated..

47. A method of restricting access to application programming interfaces on a mobile device, comprising the steps of:

loading a software application having a digital signature and a signature identification on the mobile device that requires access to one or more application programming interfaces (APIs) having at least one signature identifier;

authenticating the digital signature where the signature identification corresponds with the signature identifier; and

denying the software application access to the one or more APIs where the software application does not include an authentic digital signature .

48. The method of claim 47, wherein the digital signature and signature identification are associated with a type of mobile device.

49. The method of claim 47, comprising the additional step of:

purging the software application from the mobile device where the software application does not include an authentic digital signature. .

50. The method of claim 47, wherein:

the software application includes a plurality of digital signatures and signature identifications; and

the plurality of digital signatures and signature identifications includes digital signatures and signature identifications respectively associated with different types of mobile devices.

AMENDED SHEET

51. The method of claim 50, wherein each of the plurality of digital signatures and associated signature identifications are generated by a respective corresponding code signing authority.

52. The method of claim 47, wherein the step of determining whether the software application includes an authentic digital signature comprises the additional steps of:

verifying the authenticity of the digital signature where the signature identification corresponds with respective ones of the at least one signature identifier. .

53. The method of claim 51, wherein each of the plurality of digital signatures and signature identifications are generated by its corresponding code signing authority by applying a respective private signature key associated with the code signing authority to a hash of the software application.

54. The method of claim47, wherein the step of authenticating the digital signature where the signature identification corresponds with the signature identifier comprises the steps of:
verifying that the signature identification corresponds with the signature identifier authenticating the digital signature where signature identification corresponds with the signature identifier comprising the steps of:

storing a public signature key on a mobile device that corresponds to the private signature key associated with the code signing authority which generates the digital signature;

generating a hash of the software application to obtain a generated hash;

applying the public signature key to the digital signature to obtain a recovered hash; and

comparing the generated hash with the recovered hash.

55. The method of claim 47, wherein:

the mobile device includes a plurality of APIs;

at least one of the plurality of APIs is classified as sensitive;

access to any of the plurality of APIs requires an authentic global signature;

access to each of the plurality of sensitive APIs requires an authentic global signature and an authentic digital signature associated with a signature identification;

AMENDED SHEET

the step of determining whether the software application includes an authentic digital signature and signature identification comprises the steps of:

determining whether the one or more APIs to which the software application requires access includes a sensitive API;

determining whether the software application includes an authentic global signature; and

determining whether the software application includes an authentic digital signature and signature identification where the one or more APIs to which the software application requires access includes a sensitive API and the software application includes an authentic global signature; and

the step of denying the software application access to the one or more APIs comprises the steps of:

denying the software application access to the one or more APIs where the software application does not include an authentic global signature; and

denying the software application access to the sensitive API where the one or more APIs to which the software application requires access includes a sensitive API, the software application includes an authentic global signature, and the software application does not include an authentic digital signature and signature identifier required to access the sensitive API.

56. A code signing system for controlling access to application programming interfaces (APIs) having signature identificaters by software applications, the code signing system comprising:

a verification system for authenticating digital signatures provided by the respective software applications to access the APIs where the signature identifications correspond with the signature identificaters of the respective APIs and where a digital signature for a software application is generated with a signature identification corresponding to a signature identificater to access at least one API; and

a control system for allowing access to at least one of the APIs where the digital signature provided by the software application is authenticated by the verification system.

57. The code signing system of claim 56, wherein a virtual machine comprises the verification system and the control system.

58. The code signing system of claim 57, wherein the virtual machine is a Java virtual machine installed on a mobile device.

59. The code signing system of claim 56, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

60. The code signing system of claim 56, wherein the code signing system is installed on a mobile device and the software application is a Java application for a mobile device.

61. The code signing system of claim 56, wherein the digital signature and the signature identification of the software application are generated by a code signing authority.

62. The code signing system of claim 56, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

63. The code signing system of claim 56, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and the verification system uses a public signature key to authenticate the digital signature.

64. The code signing system of claim 63, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

AMENDED SHEET

65. The code signing system of claim 56, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

66. The code signing system of claim 56, wherein the APIs provides access to at least one of one or more core functions of a mobile device, an operating system, and hardware on a mobile device.

67. The code signing system of claim 56, wherein verification of a global digital signature provided by the software application is required for accessing any of the APIs.

68. A method of controlling access to application programming interfaces (APIs) having signature identifiers by software applications, the method comprising:

authenticating digital signatures provided by the respective software applications to access the APIs where the signature identifications correspond with the signature identifiers of the respective APIs and where a digital signature for a software application is generated with a signature identification corresponding to a signature identifier to access at least one API; and

allowing access to at least one of the APIs where the digital signature provided by the software application is authenticated.

69. The method of claim 68, wherein one digital signature and one signature identification are provided by the software application access a library of at least one of the APIs.

70. The method of claim 68, wherein the digital signature and the signature identification of the software application are generated by a code signing authority.

71. The method of claim 68, wherein the APIs access at least one of a cryptographic module that implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

AMENDED SHEET

72. The method of claim 68, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and a public signature key is used to authenticate the digital signature.

73. The method of claim 72, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the digital signature is authenticated by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

74. The method of claim 68, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

75. The method of claim 68, wherein the APIs provides access to at least one of one or more core functions of a mobile device, an operating system, and hardware on a mobile device.

76. The method of claim 68, wherein verification of a global digital signature provided by the software application is required for accessing any of the APIs

77. A management system for controlling access by software applications to application programming interfaces (APIs) having at least one signature identifier on a subset of a plurality of mobile devices, the management system comprising:

a code signing authority for providing digital signatures and signature identifications to software applications that require access to at least one of the APIs with a signature identifier on the subset of the plurality of mobile devices, where a digital signature for a software application is generated with a signature identification corresponding to a signature identifier, and the signature identifications provided to the software applications comprise those signature

AMENDED SHEET

identifications that correspond to the signature identifiers that are substantially only on the subset of the plurality of mobile devices; wherein each mobile device of the subset of the plurality of mobile devices comprises

a verification system for authenticating digital signatures provided by the respective software applications to access respective APIs where the digital identifications correspond to the digital identifiers of the respective APIs; and

a control system for allowing the respective software applications to access at least one of the APIs where the digital signatures provided by the respective software applications are authenticated by the verification system.

78. The management system of claim 77, wherein a virtual machine comprises the verification system and the control system.

79. The management system of claim 78, wherein the virtual machine is a Java virtual machine and the software applications are Java applications.

80. The management system of claim 77, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

81. The management system of claim 77, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

82. The management system of claim 77, wherein the digital signature is generated using a private signature key under a signature scheme associated with the signature identification, and the verification system uses a public signature key to authenticate the digital signature.

83. The management system of claim 82, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

84. The management system of claim 77, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

85. The management system of claim 77, wherein the subset of the plurality of mobile devices comprises mobile devices under the control of at least one of a corporation and a carrier.

86. The management system of claim 77, wherein a global digital signature provided by the software application has to be authenticated before the software application is allowed access to any of the APIs on a mobile device of the subset of the plurality of mobile devices.

87. A method of controlling access by software applications to application programming interfaces (APIs) having at least one signature identifier on a subset of a plurality of mobile devices, the method comprising:

generating digital signatures for software applications with signature identifications corresponding to respective signature identifiers of the APIs; and

providing the digital signatures and the signature identifications to software applications that require access to at least one of the APIs on the subset of the plurality of mobile devices, where the signature identifications provided to the software applications comprise those signature identifications that correspond to the signature identifiers that are substantially only on the subset of the plurality of mobile devices; wherein each mobile device of the subset of the plurality of mobile devices comprises

a verification system for authenticating digital signatures provided by the respective software applications to access respective APIs where the digital identifications correspond to the digital identifiers of the respective APIs; and

a control system for allowing the software application to access at least one of the APIs where the digital signature provided by the software application is authenticated by the verification system.

88. The method of claim 87, wherein a virtual machine comprises the verification system and the control system.

89. The method of claim 88, wherein the virtual machine is a Java virtual machine and the software applications are Java applications.

90. The method of claim 87, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

91. The method of claim 87, wherein the APIs access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

92. The method of claim 87, wherein at least one of the digital signatures is generated using a private signature key under a signature scheme associated with a signature identification, and the verification system uses a public signature keys to authenticate said at least one of the digital signatures.

93. The method of claim 92, wherein:

at least one of the digital signatures is generated by applying the private signature key to a hash of a software application under the signature scheme; and

the verification system authenticates said at least one of the digital signatures by generating a hash of the software application to obtain a generated hash, applying the public signature key to said at least one of the digital signatures to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

94. The method of claim 87, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

95. The method of claim 87, wherein the subset of the plurality of mobile devices comprises mobile devices under the control of at least one of a corporation and a carrier.

96. A mobile device for a subset of a plurality of mobile devices, the mobile device comprising:

an application platform having application programming interfaces (APIs);

a verification system for authenticating digital signatures and signature identifications provided by the respective software applications to access the APIs; and

a control system for allowing a software application to access at least one of the APIs where a digital signature provided by the software application is authenticated by the verification system;

wherein a code signing authority provides digital signatures and signature identifications to software applications that require access to at least one of the APIs such that the digital signature for the software application is generated according to a signature scheme of a signature identification, and wherein the signature identifications provided to the software applications comprise those signature identifications that are substantially only authorized to allow access on the subset of the plurality of mobile devices.

97. The mobile device of claim 96, wherein a virtual machine comprises the verification system and the control system.

98. The mobile device of claim 97, wherein the virtual machine is a Java virtual machine and the software application is a Java application.

99. The mobile device of claim 96, wherein the control system requires one digital signature and one signature identification for each library of at least one of the APIs.

AMENDED SHEET

100. The mobile device of claim 96, wherein the APIs of the application platform access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

101. The mobile device of claim 96, wherein the digital signature is generated using a private signature key under the signature scheme, and the verification system uses a public signature key to authenticate the digital signature.

102. The mobile device of claim 101, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the verification system authenticates the digital signature by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

103. The mobile device of claim 96, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

104. A method of controlling access to application programming interfaces (APIs) of an application platform of a mobile device for a subset of a plurality of mobile devices, the method comprising:

receiving digital signatures and signature identifications from software applications that require to access the APIs

authenticating the digital signatures and the signature identifications; and

allowing a software application to access at least one of the APIs where a digital signature provided by the software application is authenticated;

AMENDED SHEET

wherein a code signing authority provides the digital signatures and the signature identifications to the software applications that require access to at least one of the APIs such that the digital signature for the software application is generated according to a signature scheme of a signature identification, and wherein the signature identifications provided to the software applications comprise those signature identifications that are substantially only authorized to allow access on the subset of the plurality of mobile devices.

105. The method of claim 104, wherein one digital signature and one signature identification is required for accessing each library of at least one of the APIs.

106. The method of claim 104, wherein the APIs of the application platform access at least one of a cryptographic module, which implements cryptographic algorithms, a data store, a proprietary data model, and a user interface (UI).

107. The method of claim 104, wherein the digital signature is generated using a private signature key under the signature scheme, and a public signature key is used to authenticate the digital signature.

108. The method of claim 107, wherein:

the digital signature is generated by applying the private signature key to a hash of the software application under the signature scheme; and

the digital signature is authenticated by generating a hash of the software application to obtain a generated hash, applying the public signature key to the digital signature to obtain a recovered hash, and verifying that the generated hash with the recovered hash are the same.

109. The method of claim 104, wherein at least one of the APIs further comprises:

a description string that is displayed to a user when the software application attempts to access said at least one of the APIs.

AMENDED SHEET

(19) Europäisches Patentamt
Europ an Patent Offic
Offi e europ´ n des br vets

(11) EP 0 930 793 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
21.07.1999 Bulletin 1999/29

(51) Int Cl.[6]: **H04Q 7/32**, H04B 1/38, G06F 9/38

(21) Application number: **98310312.8**

(22) Date of filing: **16.12.1998**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **22.12.1997 US 995606**

(71) Applicant: **TEXAS INSTRUMENTS INC.**
Dallas, Texas 75243 (US)

(72) Inventors:
- **McMahon, Michael (NMI)**
  Plano, Texas 75074 (US)
- **Lineberry, Marion C.**
  Dallas, Texas 75218 (US)
- **Woolsey, Matthews A.**
  Plano, Texas 75023 (US)
- **Chauvel, Gerard (NMI)**
  06600 Antibes (FR)

(74) Representative: **Potter, Julian Mark et al**
D. Young & Co.,
21 New Fetter Lane
London EC4A 1DA (GB)

(54) Mobile equipment with a plurality of processors

(57) A wireless data platform (10) comprises a plurality of processors (12, 16). Channels of communication are set up between processors such that they may communicate information as tasks are performed. A dynamic cross compiler (80) executed on one processor compiles code into native processing code for another processor. A dynamic cross linker (82) links the compiled code for other processor. Native code may also be downloaded to the platform through use of a JAVA Bean (90) (or other language type) which encapsulates the native code. The JAVA Bean can be encrypted and digitally signed for security purposes.

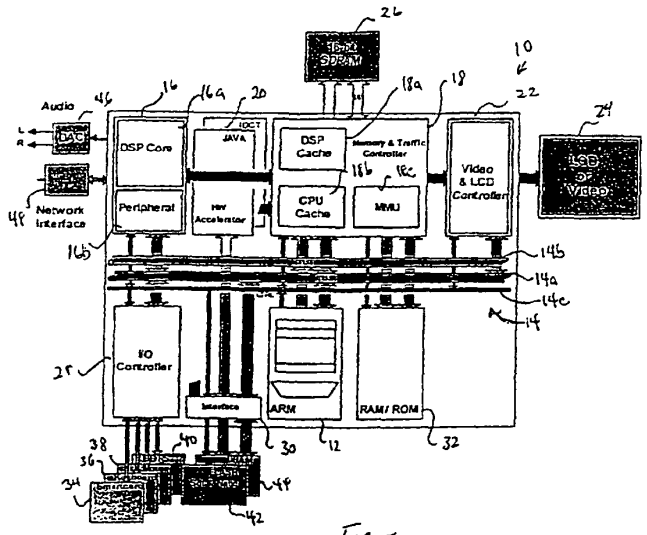


FIG 1

EP 0 930 793 A1

Description

BACKGROUND OF THE INVENTION

TECHNICAL FIELD

[0001] This invention relates in general to mobile electronic devices and, more particularly, to a hardware and software platform for mobile electronic devices.

DESCRIPTION OF THE RELATED ART

[0002] Handheld portable devices are gaining popularity as the power and, hence, functionality of the devices increases. Personal Digital Assistants (PDAs) are currently in widespread use and Smartphones, which combine some of the capabilities of a cellular phone and a PDA, are expected to have a significant impact on communications in the near future.

[0003] Some devices currently incorporate one or more DSPs (digital signal processor) or other coprocessors for providing certain discrete features, such as voice recognition, and a general purpose processor for other data processing functions. The code for the DSP and the code for the general purpose processor is generally stored in ROMs or other nonvolatile memories, which are not easily modified. Thus, as improvements and new features become available, it is often not possible to upgrade the capabilities of the device. In particular, it is not possible to maximize the use of the DSPs or other coprocessor which may be present in the device.

[0004] Therefore, a need exists for a data processing architecture which can be upgraded and optimizes use of multiple processors and coprocessors.

BRIEF SUMMARY OF THE INVENTION

[0005] The teachings of the present application disclose a mobile electronic device that comprises a coprocessor for executing native code, a host processor system operable to execute native code corresponding to the host processor system and processor independent code. The host processor system is operable to dynamically change the tasks performed by the digital signal coprocessor. Communication circuitry provides for communication between the host processor system and the coprocessor.

[0006] This mobile electronic device significant advantages over the prior art. Because the host processor system can dynamically allocate the tasks being performed by the coprocessor, which may be a digital signal processor, to fully use the coprocessor. The host processor system may direct a routine to one of a plurality of coprocessors, depending upon a variety of factors, such the present capabilities of each processor.

BRIEF DESCRIPTION OF SEVERAL VIEWS OF THE DRAWINGS

[0007] For a more complete understanding of the present invention, and the advantages thereof, reference is now made to the following descriptions taken in conjunction with the accompanying drawings, in which:

Figure 1 illustrates a block diagram of a platform architecture particularly suited for general wireless data processing;

Figure 2 illustrates a functional block diagram of the platform of Figure 1;

Figure 3 illustrates a functional block diagram of dynamic cross compiling and dynamic cross linking functions;

Figure 4 illustrate an embodiment of native code for execution on a processor being encapsulated in a JAVA Bean wrapper for downloading to a device;

Figure 5 illustrates the operation of transferring the encapsulated native code to a processor on a device from a JAVA Bean located on a remote server; and

Figure 6 illustrates a flow diagram describing security features associated with the operation of Figure 5.

DETAILED DESCRIPTION OF THE INVENTION

[0008] Figure 1 illustrates a preferred embodiment of a general wireless data platform architecture, which could be used for example, in the implementation of a Smartphone or PDA. The wireless data platform 10 includes a general purpose (Host) processor 12 coupled to bus structure 14, including data bus 14a, address bus 14b and control bus 14c. One or more DSPs (or other coprocessors) 16, including the core processor 16a and the peripheral interface 16b, are coupled to bus 14 and to memory and traffic controller 18, which includes a DSP cache memory 18a, a CPU cache 18b, and a MMU (memory management unit) 18c. Hardware accelerator circuit 20 (for accelerating a portable language such as JAVA) and a video and LCD controller 22 are also coupled to the memory and traffic controller 18. The output of the video and LCD controller is coupled to an LCD or video display 24.

[0009] Memory & traffic controller 18 is coupled to bus 14 and to the main memory 26, shown as an SDRAM (synchronous dynamic random access memory). Bus 14 is also connected to I/O controller 28, interface 30, and RAM/ROM 32. A plurality of devices could be coupled to the wireless data platform 10, such as smartcard 34, keyboard 36, mouse 38, or one or more serial ports 40, such as a USB (universal serial bus) port or an RS232 serial port. Interface 30 can couple to a flash memory card 42 and/or a DRAM card 44. The peripheral interface 16b can couple the DSP 16 to a DAC (digital to analog converter) 46, a network interface 48 or to other devices.

[0010] The wireless data platform 10 of Figure 1 utilizes both a general purpose processor 12 and a DSP 16. Unlike current devices in which the DSP 16 is dedicated to specific fixed functions, the DSP 16 of Figure 1 can be used for any number of functions. This allows the user to derive the full benefit of the DSP 16.

[0011] One main area in which the DSP 16 can be used is in connection with the man-machine interface (MMI). Importantly, functions like speech recognition, image and video compression and decompression, data encryption, text-to-speech conversion, and so on, can be performed much more efficiently using the DSP 16. The proposed architecture allows new functions and enhancements to be easily added to wireless data platform 10.

[0012] It should be noted that the wireless data platform 10 is a general block diagram and many modifications could be made. For example, Figure 1 illustrates separate DSP and processor caches 18a and 18b. As would be known to one skilled in the art, a unified cache could also be used. Further, the hardware acceleration circuit 20 is an optional item. Such devices speed the execution of languages such as JAVA; however, the circuit is not necessary for operation of the device. Further, although the illustrated embodiment shows a single DSP, multiple DSPs (or other coprocessors) could be coupled to the buses.

[0013] Figure 2 illustrates a functional software architecture for the wireless data platform 10. This block diagram presumes the use of JAVA; it should be noted that languages other than JAVA could be used as well. Functionally, the software is divided into two groups, Host processor software and DSP software. The Host software includes one or more applets 41 . The DSP API class 43 is a JAVA API package for JAVA applications or applets to access the functionality of the DSP API 50 and Host DSP Interface Layer 52. A JAVA virtual machine (VM) 45 interprets the applets. The JAVA native interface 47 is the method which the JAVA VM executes host processor or platform specific code. Native tasks 49 are non-JAVA programs which can be executed by the Host processor 12 without using the JAVA native interface. The DSP API 50, described in greater detail hereinbelow, is an API (application program interface) used the Host 12 to call to make use of the capabilities of the DSP 16. The Host-DSP Interface Layer 52 provides an API for the Host 12 and DSP 16 to communicate with each other, with other tasks, or other hardware using channels via the Host-DSP Communication Protocol. The DSP device driver 54 is the Host based device driver for the Host RTOS 56 (real time operating system) to communicate with the DSP 16. The Host RTOS 56 is an operating system, such as NUCLEUS PLUS by Accelerated Technology Incorporated.

[0014] Alternatively a non-real time operating system, such as WINDOWS CE by Microsoft Corporation, could be used. The DSP Library 58 contains programs stored for execution on the DSP 16.

[0015] On the DSP side, one or more tasks 60 can be stored in memory for execution by the DSP16. As described below, the tasks can be moved in and out of the memory as desired, such that the functionality of the DSP is dynamic, rather than static. The Host-DSP Interface layer 62 on the DSP side performs the same function as the Host-DSP Interface layer 52 on the Host side, namely it allows the Host 12 and DSP 16 to communicate. The DSP RTOS 64 is the operating system for the DSP processor. The Host Device driver 66 is a DSP based device driver for the DSP RTOS 64 to communicate with the Host 12. The Host-DSP Interface 70 couples the DSP 16 and Host 12.

[0016] In operation, the software architecture shown in Figure 2 uses the DSP 16 as a variable function device, rather than a fixed function device as in the prior art.

[0017] Accordingly, the DSP functions can be downloaded to the mobile device incorporating the architecture of Figure 2 to allow the DSP 16 to perform various signal processing functions for the Host 12.

[0018] The DSP-API provides a device independent interface from the Host 12 to the DSP 16. The functions provide the Host 12 with the ability to load and schedule tasks on the DSP 16 and to control and communicate with those tasks. The API functions include calls to: determine the DSP's available resources, create and control Host 12 and DSP tasks, create and control data channels between Host 12 and DSP tasks, and communicate with tasks. These functions are

described below. Each function returns a BOOLean result, which will be SUCCESS for a successful operation, or FAILURE. If the result is FAILURE, the errcode should be checked to determine which error occurred.

DSP_Get_MIPS

*BOOL DSP_Get_MIPS(T_DeviceID DevID, U32 *mips, U16 *errcode);*

[0019] This function returns the current MIPS available on the DSP. This consists of the MIPS capability of the DSP 16 minus a base MIPS value (the MIPS value with no additional dynamic tasks, i.e. the kernel plus API code plus drivers), minus the sum of the MIPS ratings for all loaded dynamic tasks. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING

DSP_Get_Memory_Available

*BOOL DSP_Get_Memory_Available(T_DeviceID DevID, T_Size *progmem, T_Size *datamem, U16 *errcode);*

[0020] This function will query the DSP 16 specified by *DevID* for the amounts of available memory for both program memory and data memory. The resultant values are returned in the *progmem* and *datamem* parameters. The sizes are specified in T_DSP_Words. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING

DSP_Alloc_Mem

*BOOL DSP_Alloc_Mem(T_DeviceID DevID, U16 mempage, T_Size size, T_DSP_Word **memptr, U16 *errcode);*

[0021] This function will allocate a block of memory on a DSP 16. The *DevID* specifies which device on which to allocate the memory. The *mempage* is 0 for program space, and 1 for data space. The *size* parameter specifies the memory block size in T_DSP_Words. The returned *memptr* will be a pointer to the memory block on the DSP 16, or NULL on failure. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE
DSP_NOT_ENOUGH_MEMORY

DSP_Free_Mem

*BOOL DSP_Free_Mem(T_DeviceID DevID, U16 mempage, T_DSP_Word *memptr, U16 *errcode);*

[0022] This function will free a block of memory on a DSP that was allocated with the DSP_Alloc_Mem function. The *DevID* specifies on which device the memory resides. The *mempage* is 0 for program space, and 1 for data space. The *memptr* parameter is the pointer to the memory block. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE
DSP_MEMBLOCK_NOT_FOUND

DSP_Get_Code_Info

*BOOL DSP_Get_Code_Info(char *Name, T_CodeHdr *codehdr, U16 *errcode);*

[0023] This function will access the DSP Library table and return the code header for the DSP function code specified by the *Name* parameter. On return, the location pointed to by the *codehdr* parameter will contain the code header information. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_NAMED_FUNC_NOT_FOUND

DSP_Link_Code

*BOOL DSP_Link_Code(T_DeviceID DevID, T_CodeHdr *codehdr, T_TaskCreate *tcs, U16 *errcode);*

[0024] This function will link DSP function code so that it will run at a specified address on the DSP specified by *DevID*. The *codehdr* parameter points to the code header for the function. The dynamic cross linker will link the code based on information in the code header, and in the code (COFF file). The dynamic cross linker will allocate the memory as needed, and link and load the code to the DSP 16. The *tcs* parameter is a pointer to the task creation structure needed in the DSP_Create_Task function. DSP_Link_Code will fill in the code entry points, priority, and quantum fields of the structure in preparation for creating a task. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS

DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_NOT_ENOUGH_PROG_MEMORY
DSP_NOT_ENOUGH_DATA_MEMORY
DSP_COULD_NOT_LOAD_CODE

DSP_Put_BLOB

*BOOL DSP_Put_BLOB(T_DeviceID DevID, T_HostPtr srcaddr, T_DSP_Ptr destaddr, U16 mempage, T_Size size, U16 *errcode);*

[0025] This function will copy a specified Binary Large Object (BLOB) to the DSP 16. The *DevID* specifies on which DSP 16 to copy the object. The *srcaddr* parameter is a pointer to the object in Host memory. The *destaddr* is a pointer to the location to which to copy the object on the DSP 16. The *mempage* is 0 for program space, and 1 for data space. The size parameter specifies the size of the object in T_DSP_Words. The *errcode* parameter will contain the following possible results :

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

DSP_Create_Task

*BOOL DSP_Create_Task(T_DeviceID DevID, T_TaskCreate *tcs, T_TaskID *TaskID, U16 *errcode);*

[0026] DSP_Create_Task requests the DSP 16 to create a task given the task parameters and the code locations in the DSP's program space. The Task Creation Structure is show in Table 1:

Table 1.

Task Creation Structure.		
Data Type	**Field Name**	**Description**
T_DSP_Name	Name	User defined name for the task.
U32	MIPS	MIPS used by the task.
T_ChanID	ChanIn	The channel ID used for task input.
T_ChanID	ChanOut	The channel ID used for task output
T_StrmID	StrmIn	The stream ID used for task input
T_StrmID	StrmOut	The stream ID used for task output.
U16	Priority	The task's priority.
U32	Quantum	The task's timeslice in system ticks.
T_Size	StackReq	The amount of stack required.
T_DSP_Ptr	MsgHandler	Pointer to code to handle messages to the task.
T_HOST_Ptr	CallBack	Pointer to Host code to handle messages from the task.
T_DSP_Ptr	Create	Pointer to code to execute when task is created.
T_DSP_Ptr	Start	Pointer to code to execute when task is started.
T_DSP_Ptr	Suspend	Pointer to code to execute when task is suspended.
T_DSP_Ptr	Resume	Pointer to code to execute when task is resumed.
T_DSP_Ptr	Stop	Pointer to code to execute when task is stopped.

[0027] Once the task is created, the Create entry point will be called, giving the task the opportunity to do any necessary preliminary initialization. The Create, Suspend, Resume, and Stop entry points can be NULL. The resultant *TaskID* contains both the device ID (DevID), and the DSP's task ID. If the *TaskID* is NULL, the create failed. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_PRIORITY
DSP_CHANNEL_NOT_FOUND
DSP_ALLOCATION_ERROR

DSP_Start_Task

*BOOL DSP_Start_Task(T_TaskID TaskID, U16 *errcode);*

[0028] This function will start a DSP task specified by *TaskID*. Execution will begin at the task's Start entry point. The

errcode parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

DSP_Suspend_Task

*BOOL DSP_Suspend_Task(T_TaskID TaskID, U16 *errcode);*

[0029] This function will suspend a DSP task specified by *TaskID*. Prior to being suspended, the task's Suspend entry point will be called to give the task a chance to perform any necessary housekeeping. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

DSP_Resume_Task

*BOOL DSP_Resume_Task(T_TaskID TaskID, U16 *errcode);*

[0030] This function will resume a DSP task that was suspended by DSP_Suspend_Task. Prior to being resumed, the task's Resume entry point will be called to give the task a chance to perform any necessary housekeeping. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND
DSP_TASK_NOT_SUSPENDED

DSP_Delete_Task

*BOOL DSP_Delete_Task(T_TaskID TaskID, U16 *errcode);*

[0031] This function will delete a DSP task specified by *TaskID*. Prior to the deletion, the task's Stop entry point will be called to give the task a chance to perform any necessary cleanup. This should include freeing any memory that was allocated by the task, and returning any resources the task acquired. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

DSP_Change_Task_Priority

*BOOL DSP_Change_Task_Priority(T_TaskID TaskID, U16 newpriority, U16 *oldpriority, U16 *errcode);*

[0032] This function will change the priority of a DSP task specified by *TaskID*. The priority will be changed to *newpriority*. The possible values of *newpriority* are RTOS dependent. Upon return, the *oldpriority* parameter will be set to the previous priority of the task. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND
DSP_INVALID_PRIORITY

DSP_Get_Task_Status

*BOOL DSP_Get_Task_Status(T_TaskID TaskID, U16 *status, U16 *priority, T_ChanID *Input, T_ChanID *Output, U16 *errcode);*

[0033] This function returns the status for a DSP task specified by *TaskID*. The *status* will be one of the following values:

DSP_TASK_RUNNING
DSP_TASK_SUSPENDED
DSP_TASK_WAITFOR_SEM
DSP_TASK_WAITFOR_QUEUE
DSP_TASK_WAITFOR_MSG

[0034] The *priority* parameter will contain the task's priority, and the *Input* and *Output* parameters will contain the task's input and output channel IDs, respectively. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND

DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

DSP_Get_ID_From_Nam

*BOOL DSP_Get_ID_From_Name(T_DeviceID DevID, T_DSP_Name Name, T_DSP_ID *ID, U16 *errcode);*

[0035] This function returns the ID for a named object on the DSP 16. The named object may be a channel, a task, a memory block, or any other supported named DSP object. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_NAME_NOT_FOUND

DSP_Dbg_Read_Mem

*BOOL DSP_Dbg_Read_Mem(DEVICE_ID DevID, U8 mempage, DSP_PTR addr, U32 count, DSP_WORD *buf, U16 *errcode);*

[0036] This function requests a block of memory. The *mempage* specifies program memory (0) or data memory (1). The *addr* parameter specifies the memory starting address, and the *count* indicates how many T_DSP_Words to read. The bufparameter is a pointer to a caller provided buffer to which the memory should be copied. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

DSP_Dbg_Write_Mem

*BOOL DSP_Dbg_Write_Mem(T_DeviceID DevID, U16 mempage, T DSP_Ptr addr, T_Count count, T_DSP_Word *buf, U16 *errcode);*

[0037] This function writes a block of memory. The *mempage* specifies program memory (0) or data memory (1). The *addr* parameter specifies the memory starting address, and the *count* indicates how many T_DSP_Words to write. The *buf* parameter is a pointer the buffer containing the memory to write. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

DSP_Dbg_Read_Reg

*BOOL DSP_Dbg_Read_Reg(T_DeviceID DevID, U16 RegID, T_DSP_Word *regvalue, U16 *errcode);*

[0038] This function reads a DSP register and returns the value in *regvalue.* The *RegID* parameter specifies which register to return. If the *RegID* is -1, then all of the register values are returned. The *regvalue* parameter, which is a pointer to a caller provided buffer, should point to sufficient storage to hold all of the values. The register IDs are DSP specific and will depend on a particular implementation. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_REGISTER

DSP_Dbg_Write_Reg

*BOOL DSP_Dbg_Write_Reg(T_DeviceID DevID, U16 RegID, T_DSP_Word regvalue, U16 *errcode);*

[0039] This function writes a DSP register. The *RegID* parameter specifies which register to modify. *regvalue* contains the new value to write. The register IDs are DSP specific and will depend on a particular implementation. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_REGISTER

DSP_Dbg_Set_Break

*BOOL DSP_Dbg_Set_Break(T_DeviceID DevID, DSP_Ptr addr, U16 *errcode);* This function sets a break point at the given code address *(addr).* The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND

DSP_DEVID_NOT_RESPONDING

DSP_Dbg_Clr_Break

*BOOL DSP_Dbg_Clr_Break(T_DeviceID DevID, T_DSP_Ptr addr, U16 *errcode);*

[0040] This function clears a break point that was previously set by DSP_Dbg_Set_Break at the given code address *(addr)*. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_BP_DID_NOT_EXIST

[0041] The DSP Device Driver 54 handles communications from the Host 12 to the DSP 16. The driver functions will take the communication requests as specified in the Host-DSP Communications Protocol and handle the transmission of the information via the available hardware interface. The device driver is RTOS dependent and communications hardware dependent.

[0042] The DSP Library 58 contains the blocks of code that can be downloaded to the DSP 16 for execution. Each block of code will be previously unlinked, or relocatably linked as a library, so that the dynamic cross linker can resolve all address references. Each code block will also include information about the block's requirements for DSP MIPS (millions of instructions per second), priority, time slice quantum, and memory. The format for the code block header is shown in Table 2. The program memory and data memory sizes are approximations to give the Host 12 a quick check on whether the DSP can support the task's memory requirements. If there appears to be sufficient space, the dynamic cross linker can then attempt to link and load the code. It should be noted that the dynamic cross linker could still fail, due to page alignment and contiguity requirements. In the preferred embodiment, the code is in a version 2 COFF file format.

Table 2.

Code Block Header.		
Data Type	Field Name	Description
U16	Processor	The target processor type.
T_DSP_Name	Name	Task's name.
U32	MIPS	Worst case MIPS required by the task.
T_Size	ProgSize	Total program memory size needed.
T_Size	DataSize	Total data memory size needed.
T_Size	InFrameSize	Size of a frame in the task's input channel.
T_Size	OutFrameSize	Size of a frame in the task's output channel.
T_Size	InStrmSize	Size of the task's input stream FIFO.
T_Size	OutStrmSize	Size of the task's output stream FIFO.
U16	Priority	Task's priority.
U32	Quantum	Task's time slice quantum (number of system ticks).
T_Size	StackReq	Stack required.
T_Size	CoffSize	Total size of the COFF file.
T_DSP_Ptr	MsgHandler	Offset to a message handler entry point for the task.
T_DSP_Ptr	Create	Offset to a create entry point that is called when the task is created.
T_DSP_Ptr	Start	Offset to the start of the task's code.
T_DSP_Ptr	Suspend	Offset to a suspend entry point that is called prior to the task being suspended.
T_DSP_Ptr	Resume	Offset to a resume entry point that is called prior to the task being resumed.
T_DSP_Ptr	Stop	Offset to a stop entry point that is called prior to the task being deleted.
T_Host_Ptr	CoffPtr	Pointer to the location of the COFF data in the DSP Library.

[0043] A procedure for converting portable (processor independent) code, such as JAVA code, into linked target code is shown in Figure 3. The procedure uses two functions, a dynamic cross compiler 80 and a dynamic cross linker 82. Each function is implemented on the host processor 12. The dynamic cross linker is part of the DSP-API in the preferred embodiment. The cross compiler may also be part of the DSP-API.

[0044] The dynamic cross compiler 80 converts portable code into unlinked, executable target processor code. The dynamic cross linker 82 converts the unlinked, executable target processor code into linked, executable target processor code. To do so, it must resolve addresses within a block of code, prior to loading on the DSP 16. The dynamic

cross linker 82 links the code segments and data segments of the function, allocates the memory on the DSP 16, and loads the code and constant data to the DSP 16. The functions are referred to as "cross" compiling and "cross" linking, because the functions (compiling and linking) occur on a different processor (i.e., the host processor 12) from the target processor which executes the code (i.e., the DSP 16).

[0045] The dynamic cross compiler 80 accepts previously unlinked code loaded on demand by a user or a user agent (such as a browser). The code is processed to either (1) identify "tagged" sections of the code or (2) analyze untagged code segments for suitability of execution on the DSP 16. A tagged section of source code could delineate source targetable to a DSP by predetermined markers such as "<start DSP code>" and <end DSP code>" embedded in the source code. If a tagged section is identified either directly or through analysis, a decision is made to either cross compile or not based on the current processing state of the DSP 16. If a decision is made to compile, the section of code processed by compiling software that outputs unlinked, executable target processor code, using well known compiling methods. A decision not to compile could be made if for example, the DSP has insufficient available processing capacity (generally stated as available MIPS - million of instructions per second) or insufficient available memory, due to other tasks being executed by the DSP 16. The compiled code can be passed to the dynamic cross linker 82 for immediate use in the DSP 16, or could be saved in the DSP library 58.

[0046] The dynamic cross linker 82 accepts previously unlinked code, which is either (1) statically stored in connection with the host processor 12 or (2) dynamically downloaded to the host processor 12 over a network connection (including global networks such as the Internet) or (3) dynamically generated by the dynamic cross compiler 80. The dynamic cross linker 82 links the input code for a memory starting address of the DSP 16 determined at runtime. The memory starting address can be determined from a memory map or memory table stored on and managed by either the host processor 12 or DSP 16. The dynamic cross linker 82 convert referenced memory locations in the code to actual memory locations in the DSP 16. These memory locations could include, for example, branch addresses in the code or references to locations of data in the code.

[0047] In the preferred embodiment, the portable code is in a COFF (common object file format) which contains all information about the code, including whether it is linked or unlinked. If it is unlinked, symbol tables define the address which must be changed for linking the code.

[0048] The conversion process described above has several significant advantages over the prior art. First, the dynamic cross compiler 80 allows run-time decisions to be made about where to execute the downloaded portable code. For example, in a system with multiple target processors (such as two DSPs 16), the dynamic cross compiler 80 could compile the portable code to any one of the target processors based on available resources or capabilities. The dynamic cross linker 82 provides for linking code to run on a target processor which does not support relocatable code. Since the code is linked at run-time, memory locations in the DSP 16 (or other target processor) do not need to be reserved, allowing optimum efficiency of use of all computing resources in the device. Because the compiling is accomplished with knowledge of the architecture of the platform 10, the compiling can take advantage of processor and platform specific features, such as intelligent cache architectures in one or both processors 12 and 16.

[0049] Thus, the DSP 16 can have various functions which are changed dynamically to fully use its processing capabilities. For example, the user may wish to 12 load a user interface including voice recognition. At that time, the host processor 12 could download software and dynamically cross compile and cross link the voice recognition software for execution in the DSP 16. Alternatively, previously compiled software in the DSP library 58 could be dynamically cross linked, based on the current status of the DSP 16, for execution.

[0050] The Host Device Driver handles communications from the DSP 16 to the Host Processor 12. The driver functions takes the communication requests as specified in the Host-DSP Communications Protocol and handles transmission of the information via the available hardware interface. The device driver is RTOS dependent and communications hardware dependent.

[0051] The Host-DSP Communications Protocol governs the communications of commands and data between the Host 12 and the DSP 16. The communications consist of several paths: messages, data channels, and streams. Messages are used to send initialization parameters and commands to the tasks. Data channels carry large amounts of data between tasks and between the DSP 16 and Host 12, in the form of data frames. Streams are used to pass streamed data between tasks and between the DSP 16 and Host 12.

[0052] Each task has an entry point to a message handler, which handles messages. The messages are user defined and will include initialization parameters for the task's function, and commands for controlling the task. The tasks send messages to the Host 12 via the callback specified when the task is created. The prototype for the task's message handler and the prototype for the Host's callback are shown here:

```
void TaskMsgHandler(T_ReplyRef replyref, T_MsgID MsgID, T_Count count, T_DSP_Word *buf);
void HostCallBack(T_ReplyRef replyref T_MsgID MsgID, T_Count count, T_DSP_Word *buf);
```

[0053] The *replyref* parameter refers to an implementation dependent reference value, which is used to route the

reply back to the sender. For every Send_Message call, the recipient must call Reply_Message using the *replyref* parameter. The actual messages may appear as follows:

Sent message	MsgPktFlag	taskid	replyref	msgid	count	buf[.....]
Reply message	MsgPktFlag	-1	replyref	msgid	count	buf[.....]

The multiword data is sent least-significant word first.

[0054] A *TaskID* of 0 in the Send_Message function indicates a system level message. The system level messages are used to implement the DSP-API functions

[0055] Following are the Message functions:

Send_Message

*BOOL Send_Message(T_TaskID TaskID, T_MsgID MsgID, T_Count count, T_DSP_Word *msgbuf, T_DSP_Word *replybuf, T_Size replybufsize, T_Count replycount, U16 *errcode);*

[0056] This function will send a user defined message to a task specified by *TaskID.* The *MsgID* defines the message, and the *msgbufcontains* the actual message data. The message size is *count* T_DSP_Words. The reply *to* the message will be contained in the *replybuf* parameter, which points to a buffer of size *replybufsize,* provided by the caller. It should be of sufficient size to handle the reply for the particular message. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

Reply_Message

*BOOL Reply_Message(T_ReplyRef replyref, T_Count count, T_DSP_Word *buf, U16 *errcode);*

[0057] This function is used to reply to messages. The *replyref* parameter is a reference used to route the reply back to the sender of the original message, and is implementation specific. The reply is contained in the *buf* parameter and its size is *count* T_DSP_Words. The *errcode* parameter will contain the following possible results:

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_BAD_REPLY_REF

[0058] The concept of channels is used to transmit frame-based data from one processor to another, or between tasks on the same processor. When created, a channel allocates a specified number and size of frames to contain the data. Initially, the channel will contain a list of empty frames. Tasks that produce data will request empty frames in which to put the data, then once filled, the frame is returned to the channel. Tasks that consume data will request full frames from the channel, and once emptied, the frame is returned to the channel. This requesting and returning of frame buffers allows data to move about with a minimum of copying.

[0059] Each task has a specified Input and Output channel. Once a channel is created, it should be designated as the input to one task, and the output to another task. A channel's ID includes a device ID, so channels can pass data between processors. Channel data flow across the Host-DSP interface may appear as follows:

ChanPktFlag	Channel ID	Count	Data[...]

The following are the channel functions:

Create_Channel

*BOOL Create_Channel(T_DeviceID DevID, T_Size framesize, T_Count numframes, T_ChanID *ChannelID, U16 *errcode);*

[0060] This function creates a data frame-based communication channel. This creates a channel control structure, which maintains control of a set of frame buffers, whose count and size are specified in the *numframes* and *framesize* parameters, respectively. When created, the channel allocates the data frames, and adds them to its list of empty frames. *ChannelID* will return the ID of the new channel. If the *DevID* is not that of the calling processor, a channel control structure is created on both the calling processor and the *DevID* processor, to control data flowing across the communications interface. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING

CHAN_ALLOCATION_ERROR

Del te_Channel

*BOOL Delete_Channel(T_ChanID ChannelID, U16 *errcode);*

[0061] This function deletes an existing channel specified by *ChannelID*. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT RESPONDING
CHAN_CHANNEL_NOT_FOUND

Request_Empty_Frame

*BOOL Request_Empty_Frame(T_LocalChanID Chn, T_DSP_Word **bufptr, BOOL WaitFlag, U16 *errcode);*

[0062] This function requests an empty frame from the specified local channel ID. If *Chn* is NULL, then the task's output channel is used. Upon return, the *bufptr* parameter will contain the pointer to the frame buffer. If the *WaitFlag* is TRUE, and there is no frame buffer available, the caller will be suspended until a buffer becomes available. If the *WaitFlag* is FALSE, the function will return regardless. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_UNAVAILABLE

Return_Full_Frame

*BOOL Return_Full_Frame(T_LocalChanID Chn, T_DSP_Word *bufptr, U16 *errcode);*

[0063] Once a task has filled a frame buffer, it returns is to the channel using this function. The buffer pointed to by *bufptr* is returned to the channel ID specified. If *Chn* is NULL, then the task's output channel is used. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_CTRL_ERROR

Request_Full_Frame

*BOOL Request_Full_Frame(T_LocalChanID Chn, T_DSP_Word **bufptr, BOOL WaitFlag, U16 *errcode);*

[0064] This function requests a full frame of data from the specified local channel ID. If *Chn* is NULL, then the task's input channel is used. Upon return, the *bufptr* parameter will contain the pointer to the frame buffer. If the *WaitFlag* is TRUE, and there are no full frame buffers available, the caller will be suspended until a buffer becomes available. If the *WaitFlag* is FALSE, the function will return regardless. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_UNAVAILABLE

Return_Empty_Frame

*BOOL Return_Empty_Frame(T_LocalChanID Chn, T_DSP_Word *bufptr, U16 *errcode);*

[0065] Once a task has used the data from a frame buffer, it should return the buffer to the channel using this function. The buffer pointed to by *bufptr* is returned to the channel ID specified. If *Chn* is NULL, then the task's input channel is used. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_CTRL_ERROR

Set_Task_Input_Channel

*BOOL Set_Task_Input_Channel(T_Task *TaskID, T_ChanID ChanID, U16 *errcode);*

[0066] This function sets a task's input channel to the specified channel ID. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_TASK_NOT_FOUND
CHAN_CHANNEL_NOT_FOUND

Set_Task_Output_Channel

*BOOL Set_Task_Output_Channel(T_Task *TaskID, T_ChanID ChanID, U16 *errcode);*

[0067] This function sets a task's output channel to the specified channel ID. The *errcode* parameter will contain the following possible results:

CHAN_SUCCESS

CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_TASK_NOT_FOUND
CHAN_CHANNEL_NOT_FOUND

[0068] Streams are used for data, which can not be broken into frames, but which continuously flow into and out of a task. A stream will consist of a circular buffer (FIFO) with associated head and tail pointers to track the data as it flows in and out. Each task can have a designated input and output stream. Stream data flow across the Host-DSP interface may appear as follows:

StrmPktFlag	Stream ID	Count	Data[...]

Following are the stream functions:

Create_Stream

*BOOL Create_Stream(T_DeviceID DevID, T_Size FIFOsize, T_StrmID *StreamID, U16*errcode);*

[0069] This function creates a FIFO-based communication stream. This creates a stream control structure, which maintains control of a FIFO of size *FIFOsize*. When created, the stream allocates an empty FIFO, and initializes head and tail pointers to handle data flow into and out of the stream. *StreamID* will return the ID of the new stream. If the *DevID* is not that of the calling processor, a stream control structure is created on both the calling processor and the *DevID* processor, to control data flowing across the communications interface. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_ALLOCATION_ERROR

Delete_Channel

*BOOL Delete_Stream(T_StrmID StreamID, U16 *errcode);*

[0070] This function deletes an existing stream specified by *StreamID*. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND

Get_Stream_Count

*BOOL Get_Stream_Count(T_LocalStrmID StrmID, T_Count *count, U16 *errcode);*

[0071] This function requests the count of T_DSP_Words currently in the stream FIFO specified by *StrmID*. The *count* parameter will contain the number upon return. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_STREAM_NOT_FOUND

Write_Stream

*BOOL Write_Stream(T_LocalStrmID Strm, T_DSP_Word *bufptr, T_Count count, T_Count *countwritten, U16 *errcode);*

[0072] This function will write *count* number of T_DSP_Words to the stream specified by the *Strm*. If *Strm* is NULL, the task's output stream is used. The data is pointed to by the *bufptr* parameter. Upon return, *countwritten* will contain the number of T_DSP_Words actually written. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND
STRM_STREAM_OVERFLOW

Read_Stream

*BOOL Read_Stream(T_LocalStrmID Strm, T_DSP_Word *bufptr, T_Count maxcount, BOOL WaitFlag, T_Count *countread, U16 *errcode);*

[0073] This function reads data from the stream specified by *Strm*. If *Strm* is NULL, the task's input stream is used. The data will be stored in the buffer pointed to by *bufptr*. Up to *maxcount* T_DSP_Words will be read from the stream. The *countread* parameter will contain the actual count of the data read. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS

STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND

S t_Task_Input_Stream

*BOOL Set_Task_Input_Stream(T_Task *TaskID, T_StrmID StrmID, U16 *errcode);*

[0074] This function sets a task's input stream to the specified stream ID. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_TASK_NOT_FOUND
STRM_STREAM_NOT_FOUND

Set_Task_Output_Stream

*BOOL Set_Task_Output_Stream(T_Task *TaskID, T_StrmID StrmID, U16 *errcode);*

[0075] This function sets a task's output stream to the specified stream ID. The *errcode* parameter will contain the following possible results:

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_TASK_NOT_FOUND
STRM_STREAM_NOT_FOUND

[0076] Data types used herein are defined in Table 3:

Table 3

Symbol	Description
S8	Signed 8-bit integer.
U8	Unsigned 8-bit integer.
S16	Signed 16-bit integer.
U16	Unsigned 16-bit integer.
S32	Signed 32-bit integer.
U32	Unsigned 32-bit integer.
T_HostWord	A word on the Host processor.
T_DSP_Word	A word on the DSP processor.
BOOL	Boolean value (TRUE or FALSE).
T_HostPtr	Pointer on the Host processor.
T_DSP_Ptr	Pointer on the DSP processor.
T_DeviceID	Processor device ID.
T_TaskID	A structure containing fields for a device ID and a processor local task ID.
T_ChanID	A structure containing fields for a device ID and a processor local channel ID.
T_MsgID	Message ID.
T_DSP_ID	An object ID on the DSP.
T_Count	Data type for a count.
T_Size	Data type for a size.
T_HostCallBack	Value used when tasks send message back to the Host.
T_ReplyRef	Message reply reference.
T_LocalTaskID	Local task ID.
T_LocalChanID	Local channel ID.

Table 3 (continued)

Symbol	Description
T_DSP_Name	Name for DSP objects (RTOS dependent).
T_CodeHdr	Code header structure for a DSP Library entry.
T_TaskCreate	Task creation structure.

[0077] These tables define the messages passing between devices (i.e. Host to DSP 16). The device IDs present as parameters in the corresponding function calls are not incorporated in the messages since they are used to actually route the message to the device. Similarly, task IDs that include a device ID as their upper half for the function call will not include the device ID in the message, but only the DSP's local task ID portion.

Table 4

DSP-API Messages			
Message	Send Parameters	Reply Parameters	Direction Host ⇔ DSP
GET_MIPS	None	U32 mips	→
GET_MEM_AVAIL		T_Size progmem T_Size datamem	→
ALLOC_MEM	U16 mempage T_Size size	T_DSP_Word *memptr U16 errcode	→
FREE_MEM	U16 mempage T_DSP_Word *memptr	U16 errcode	→
PUT_BLOB	T_DSP_Ptr destaddr U16 mempage T_Size size T_DSP_Word BLOB[size]	U16 errcode	→
CREATE_TASK	T_TaskCreate tcs	T_TaskID TaskID U16 errcode	→
START_TASK	T_TaskID TaskID	U16 errcode	→
SUSPEND_TASK	T_TaskID TaskID	U16 errcode	→
RESUME_TASK	T_TaskID TaskID	U16 errcode	→
DELETE_TASK	T_TaskID TaskID	U16 errcode	→
CHANGE_PRIORIT Y	T_TaskID TaskID U16 newpriority	U16 oldpriority U16 errcode	→
GET_TASK_STATUS	T_TaskID TaskID	U16 status U16 priority T_ChanID Input T_ChanID Output U16 errcode	→
GET_ID	T_DSP_Name Name	T_DSP_ID ID U16 errcode	→

Table 5

DSP Interface Layer / Channel Interface Layer Messages			
Message	Send Parameters	Reply Parameters	Direction Host ⇔ DSP
CREATE_CHANNEL	T_Size framesize T_Count numframes	T_ChanID ChannelID U16 errcode	→

Table 5 (continued)

DSP Interface Layer / Chann I Interface Layer Messages			
Message	Send Parameters	Reply Parameters	Direction Host ⇔ DSP
DELETE_CHANNEL	T_ChanID ChannelID	U16 errcode	→
CREATE_STREAM	T_Size FIFOsize	T_StrmID StreamID U16 errcode	→
DELETE_STREAM	I_StrmID StreamID	U16 errcode	→

Table 6

Debug Messages			
Message	Send Parameters	Reply Parameters	Direction Host ⇔ DSP
READ_MEM	U16 mempage T_DSP_Ptr addr T_Count count	T_DSP_Word mem[count] U16 errcode	→
WRITE_MEM	U16 mempage T_DSP_Ptr addr	U16 errcode	→
	T_Count count T_DSP_Word mem[count]		
READ_REG	U16 RegID	DSP_WORD regvalue U16 errcode	→
WRITE_REG	U16 RegID T_DSP_Word regvalue	U16 errcode	→
SET_BREAK	T_DSP_Ptr addr	U16 errcode	→
CLR_BREAK	T_DSP_Ptr addr	U16 errcode	→
BREAK_HIT	T_DSP_Ptr addr	U16 ACK	←

[0078] Figures 4 - 6 illustrate an embodiment for downloading native code to a target processor (i.e., the host 12 or DSP 16) in a secure and efficient manner. This embodiment for downloading code could be used, for example, in downloading code from the Internet, or other global network, from a Local or Wide Area Network, or from a peripheral device, such as a PC Card or Smartcard.

[0079] In Figure 4, an embodiment of a JAVA Bean 90 is shown, where the Bean 90 acts as a wrapper for native code 92. The Bean further includes several attributes 94, listed as a Code Type attribute 94a, a Code Size attribute 94b and a MIPS Required attribute 94c. The Bean 90 has several actions 96, including a Load Code action 96a, a Load Parameters action 96b and an Execute Parameter 96c.

[0080] In operation, the Load Code action 96a is used to load external native code (native to the target processor) into the Bean. Since JAVA Beans have persistence, the Bean 90 can store its internal state, including the native code 92 and the attributes 94. The Load Parameters action 96b retrieves parameters from the native code 92 (using, for example, the COFF file format described above) and stores the parameters as attributes 94a-c. The Execute action 96c executes tasks installed in the DSP 16.

[0081] Figure 5 illustrates use of the Bean 90 to download code to the target processor. In this example, it is assumed that the target processor is the DSP 16 (or one of multiple DSPs 16), although it could be used to download native code to the host processor 12 as well. Further, it is assumed that the desired Bean 90 is resident in a network server, such as a LAN server or an Internet server, although the Bean could be resident in any device in communication with the platform 10, such as a Smartcard. For a wireless data platform 10, the connection to the network server 100 will often be wireless.

[0082] In Figure 5, the platform 10 is coupled to a network server 100. The host processor 12, as shown in greater detail in Figure 2, may execute one or more JAVA applets 41 through a JAVA virtual machine 45. In order to download new code, the host 12 loads an applet 41 containing the Bean 90 from the network server 100 or the Bean, without the containing applet, can be downloaded from the server 100. Once the wrapper Bean 90 has been retrieved, it can be queried for the size of the native code, code type (for which processor is the code intended) and MIPs required. If

the intended processor has sufficient resources to run the code 92, the code 92 can be installed to execute on the intended processor, either the host processor 12 or DSP 16 in the architecture shown in Figure 5. Typically, the native code 92 will be unlinked, compiled code. Thus, the cross linker 82 of the DSP-API 50 will link the code to an available memory location. The Bean would pass the binary native code 92 to the dynamic cross linker 82, which would install and execute the code.

[0083] A typical manner in which a download of native code might occur is when the user is running an applet 41 in which a DSP function is desired. First, the applet 41 would check to see if the desired code was installed as a task 60 in the DSP or was available in the DSP Library 58. If so, the task could be executed without a download.

[0084] If the task is not stored in the DSP 16 or the DSP library 58, an object (referred to as the " DSPLoader" object herein) could be created to load the Bean 90. If the DSPLoader class is local on the host 12, JAVA will check to see if the Bean is available locally as well. In a first instance, there may be a Bean with the code stored locally. If so, the code from the Bean is installed to the DSP 16 (or to whichever processor specified by the Code Type). If a Bean without the code is stored locally, the Bean can retrieve the code from the appropriate server.

[0085] On the other hand, if the DSPLoader object is not local, then JAVA will load the Bean 90 from the server that wrote the applet 41. The code from the Bean will then be installed as described above.

[0086] While the downloading of native code is described in connection with the use of a JAVA Bean, it could also be accomplished by wrapping the code within another language, such as an ActiveX applet.

[0087] Using a JAVA Bean (or other applet) as a wrapper to the native code has significant advantages. First, it allows a simple, standard method for loading code onto one of a plurality of processors. The Bean is created, code is loaded into the Bean and the code is linked to the appropriate processor. Without wrapping the code within the Bean, the process may take several hundred steps. Second, it allows multiple pieces of native code to be combined by a single applet, providing for complex applications to be generated from multiple discrete routines using a single applet to combine the routines as desired. Third, it takes advantage of the language's security features, thereby protecting not only the JAVA code in the Bean 90, but the native code 92 as well. Other languages, such as ActiveX, have security features as well.

[0088] Two of the most important security features are digital signing and encryption. A JAVA Bean or ActiveX applet may be signed by the source of the code; when the Bean or applet is downloaded, the signature is verified by the receiving application, which has a list of trusted sources. If the Bean or applet is signed by a trusted source, it can be decrypted using standard techniques. Accordingly, the native code is encrypted during transmission along with the code of the Bean or applet, preventing unauthorized modification of the code. Because the native code is secure and comes from a trusted source, the attributes can also be trusted as accurate.

[0089] Figure 6 illustrates a flow chart describing the process of downloading native code for a processor using a JAVA Bean, it being understood that the native code could be wrapped in an applet of a different language using similar techniques. In step 110, the encrypted, digitally signed Bean 90 is downloaded to a device running a JAVA virtual machine. In step 112, the signature is verified. If it is not from a source listed as a trusted source, exception processing is enabled in step 114. In the case of the Bean coming from a trusted source, the exception processing function may give the user an opportunity to accept the Bean, if the user is comfortable with the source. If the signature is invalid, the exception processing may delete the Bean 90 and send an appropriate error message to the user.

[0090] If the signature is valid and comes from a trusted source, the Bean is decrypted in step 116. This step decrypts both the JAVA code and the native code in the Bean. In step 118, the attributes are retrieved from the Bean 90 and in step 120 the applet determines whether the appropriate processor has sufficient resources to run the code. If not, the exception processing step 114 may decline to install the native code, or steps may be taken to free resources. If there are sufficient resources, the code is linked using the cross-linker and installed on the desired processor in step 122. In step 124, the native code is executed.

[0091] Sample JAVA script for a Bean 90 is provided hereinbelow:

```
package ti.dsp.loader;

import java.awt.*;
import java.io.*;
import java.net.*;

public class NativeBean extends Canvas implements Serializable
{
    public NativeBean() {

        setBackground(Color.white);

        funcData = new ByteArrayOutputStream();

        try {
            funcCodeBase = new URL("http://localhost");
        }
        catch (MalformedURLException e) {
```


BNSDOCID: <EP___0930793A1_I_>

```
    }
}

public Dimension getMinimumSize() {

    return new Dimension(50, 50);
}

public void loadCode() {

    URL baseURL = null;

    try {
        baseURL = new URL(funcCodeBase.toString() + "/" + myFunction);
    }
    catch (MalformedURLException e) {
    }

    DataInputStream source = null;
    int read;
    byte[] buffer;

    buffer = new byte[1024];
    try {
        source = new DataInputStream(baseURL.openStream());
    }
    catch (IOException e) {
        System.out.println("IOException creating streams: " + e);
    }

    codeSize = 0;

    funcData.reset();

    try {
        while (true) {

            read = source.read(buffer);

            if (read == -1)
                break;

            funcData.write(buffer, 0, read);
        }
    }
    catch (IOException e) {
        System.out.println("IOException: " + e);
    }
```

```
        codeSize = funcData.size();
        System.out.println("Code size = " + codeSize);

        try {
            source.close();
        }
        catch (IOException e) {
            System.out.println("IOException closing: " + e);
        }
    }

    public synchronized String getFunctionName() {

        return myFunction;
    }

    public void setFunctionName(String function) {

        myFunction = function;
    }

    public synchronized String getCodeBase() {

        return funcCodeBase.toString();
    }

    public void setCodeBase(String newBase) {

        try {
            funcCodeBase = new URL(newBase);
        }
        catch (MalformedURLException e) {
        }
    }

    public void installCode() {

        FileOutputStream destination = null;
        File libFile = new File(myFunction);

        try {
            destination = new FileOutputStream(libFile);
        }
        catch (IOException e) {
            System.out.println("IOException creating streams: " + e);
        }

        if (destination != null) {
```

```
                try {
                    funcData.writeTo(destination);
                }
                catch (IOException e) {
                    System.out.println("IO Exception installing native code: " + e);
                }
            }
        }
            linkCode(funcData)

        public void loadParameters() {
        }

        public void execute() {
        }

        public synchronized int getCodeSize() {

            return codeSize;
        }

        public synchronized int getCodeType() {

            return codeType;
        }

        public void setCodeType(int newType) {

            codeType = newType;
        }

        private int codeSize = 0;
        private int codeType = 1;
        private String myFunction = "";
        private URL funcCodeBase = null;
        private ByteArrayOutputStream funcData = null;
    }
```

[0092] In the script set forth above, the NativeBean() routine creates the Bean 90 which will hold the native code. The loadCode() routine gets the native code from the server. The getFunctionName() and getCodeBase() routines retrieve attributes. The installCode() routine calls the cross linker to link the native code to the DSP and to load the linked code. The loadParameters() routine instructs the Bean to examine the native code and determine its attributes. The getCodesize() and getCodetype() routines transfer the attributes to the requesting applet.

[0093] Although the teachings disclosed herein have been directed to certain exemplary embodiments, various modifications of these embodiments, as well as alternative embodiments, will be suggested to those skilled in the art.

[0094] Further and particular embodiments of the invention will now be enumerated with reference to the following numbered clauses.

1. A mobile electronic device, comprising:

   a coprocessor for executing native code;
   a host processor system operable to execute native code corresponding to the host processor system and processor independent code, said host processor system operable to dynamically change the tasks performed

by the digital signal coprocessor; and
circuitry for communicating between said host processor system and said coprocessor.

2. The mobile electronic device of clause 1 and further comprising network interface circuitry for receiving data from a network.

3. The mobile electronic device of clause 2 wherein said network interface circuitry comprises wireless network circuitry.

4. The mobile electronic device of clause 3 wherein said network interface circuitry comprises circuitry for interfacing with a global network.

5. A method of controlling a mobile electronic device comprising *the* steps of:

executing native code in a coprocessor;
executing both native code and processor independent code in a host processor system;
dynamically changing the tasks performed by the digital signal coprocessor with said host processor system; and
communicating between said host processor system and said coprocessor.

6. The method of clause 5 and further comprising the step of receiving code through a network interface.

7. The method of clause 6 and further comprising the step of receiving code through a wireless network interface.

8. The method of clause 6 or 7 and further comprising the step of receiving code through a wireless network interface from a global network.

9. A mobile electronic device, comprising:

a plurality of coprocessors;
a host processor system operable to:

execute source code;
identify one or more sections of source code to be executed on one or more of said coprocessors; and
for each identified section of source code, determining a corresponding coprocessor; and
for each identified section of source code, compile said identified section of code into the native code associated with said corresponding coprocessor and install said native code onto said corresponding coprocessor; and

circuitry for communicating between said host processor system and said coprocessors.

10. The mobile electronic device of clause 9 wherein one or more of said coprocessors comprise digital signal processors.

Claims

1. A mobile electronic device, comprising:

a coprocessor for executing native code;
a host processor operable to execute native code corresponding to the host processor and processor independent code, said host processor operable to dynamically change the tasks performed by the digital signal coprocessor; and
circuitry for communicating between said host processor and said coprocessor.

2. The mobile electronic device of Claim 1, wherein said coprocessor comprises a digital signal processor.

3. The mobile electronic device of Claim 1 or Claim 2, wherein said processor independent code comprises JAVA.

4. The mobile electronic device of any preceding claim , wherein said host processor system is arranged to generate native code for said coprocessor.

5. The mobile electronic device of any preceding claim, wherein said host processor is arranged to generate native code for said coprocessor by compiling processor independent source code.

6. The mobile electronic device of any preceding claim, wherein said host processor is arranged to compile identified blocks of source code.

7. The mobile electronic device of any preceding claim, wherein said host processor system is arranged to identify blocks of source code that could be executed on the coprocessor and to compile said blocks of code.

8. The mobile electronic device of any preceding claims, further comprising:
a memory for storing a library of routines that can be downloaded to said coprocessor for execution.

9. The mobile electronic device of any preceding claim further comprising a hardware language accelerator.

10. The mobile electronic device of any preceding claim wherein said hardware accelerator comprises a JAVA accelerator.

11. The mobile electronic device of any preceding claim further comprising network interface circuitry for receiving data from a network.

12. A method of controlling a mobile electronic device comprising of:

executing native code in a coprocessor;
executing both native code and processor independent code in a host processor
dynamically changing the tasks performed by the digital signal coprocessor with said host processor and communicating between said host processor system and said coprocessor.

13. The method of claim 12 wherein said step of executing native code in a coprocessor comprises executing native code in a digital signal processor.

14. The method of claims 12 and 13 further comprising generating native code for coprocessor in said general processing system.

15. The method of claim 14 wherein said step of generating native code comprises the step of generating native code by compiling processor independent source code.

16. The method of any of claims 12 to 15 further comprising identifying blocks of said source code to compile for execution on said coprocessor.

17. The method of any of claims 12-16 further comprising storing a library of routines for downloading from said host processor system to said coprocessor for execution.

18. A mobile electronic device, comprising:

a plurality of coprocessors;
a host processor system operable to:

execute source code;
identify one or more portions of source code to be executed on one or more of said coprocessors; and
for each identified portion of source code, determining a corresponding coprocessor; and
for each identified portion of source code, compile said identified portion of code into the native cod associated with said corresponding coprocessor and install said native code onto said corresponding coprocessor; and

circuitry for communicating between said host processor system and said coprocessors.

19. A method of controlling a mobile electronic device, comprising:

executing source code on a host processor system;
identifying one or more portions of source code to be executed on one or more coprocessors; and
for each identified portion of source code, determining a corresponding coprocessor; and
for each identified portion of source code, compiling said identified portion of code into the native code associated with said corresponding coprocessor and installing said native code onto said corresponding coprocessor; and
communicating between said host processor system and said coprocessors.

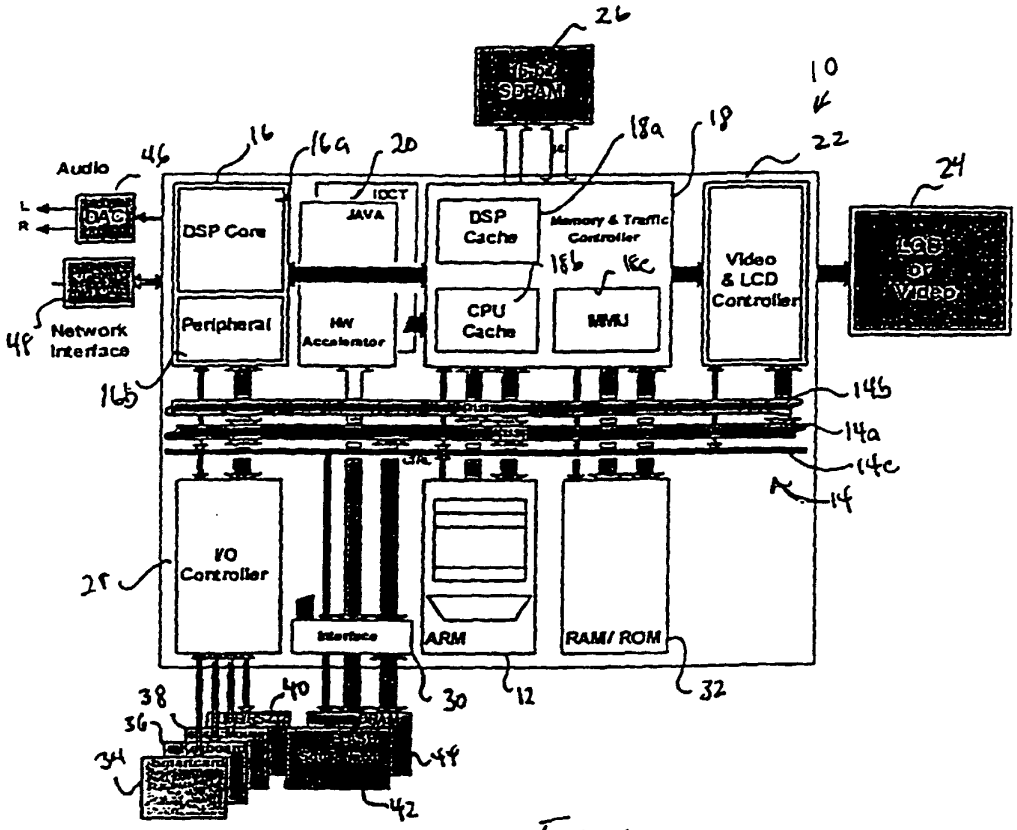


FIG 1

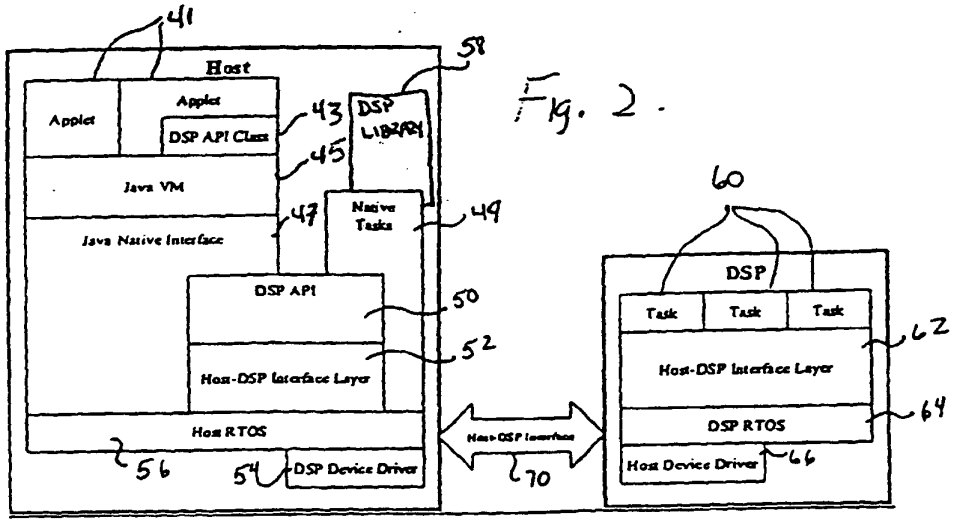


Fig. 2.

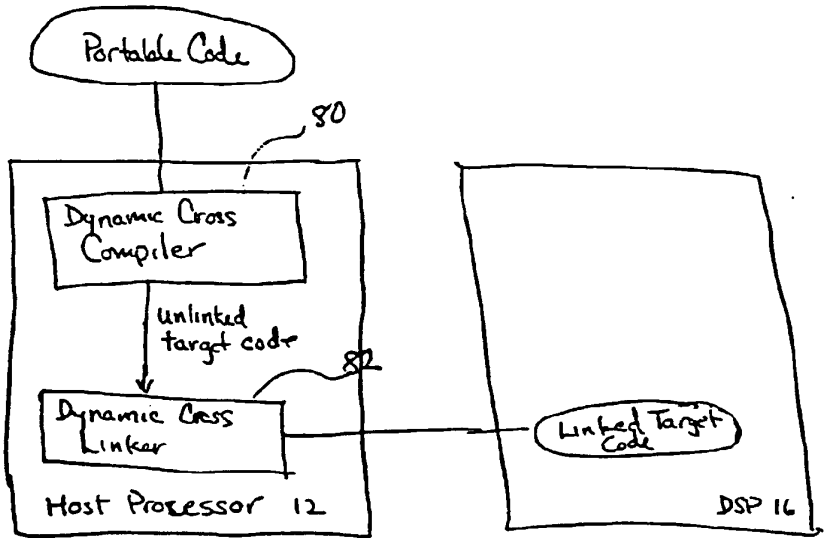


Figure 3

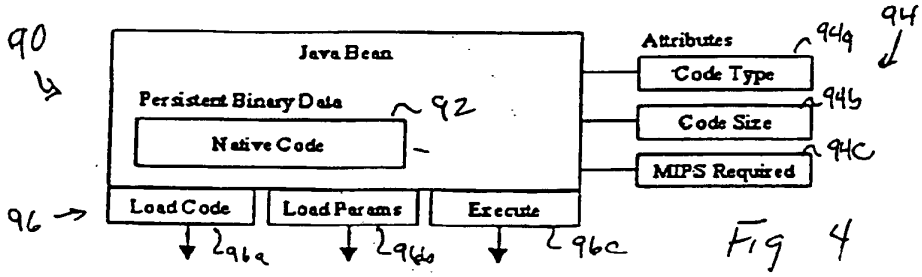


Fig 4

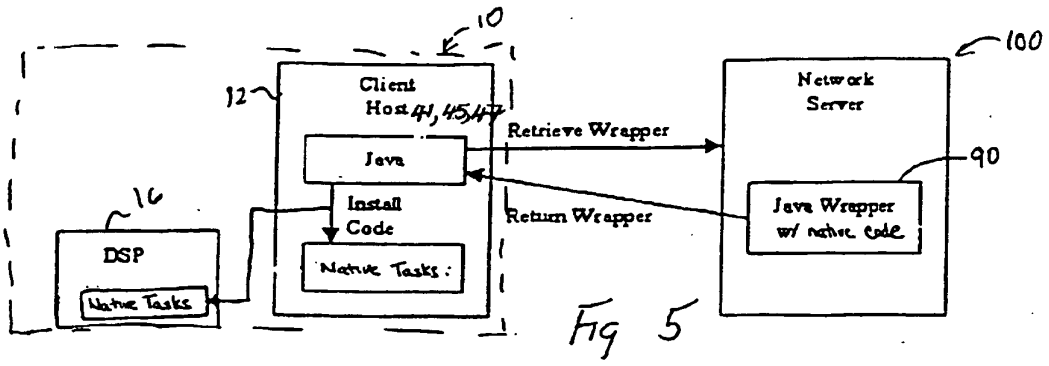


Fig 5

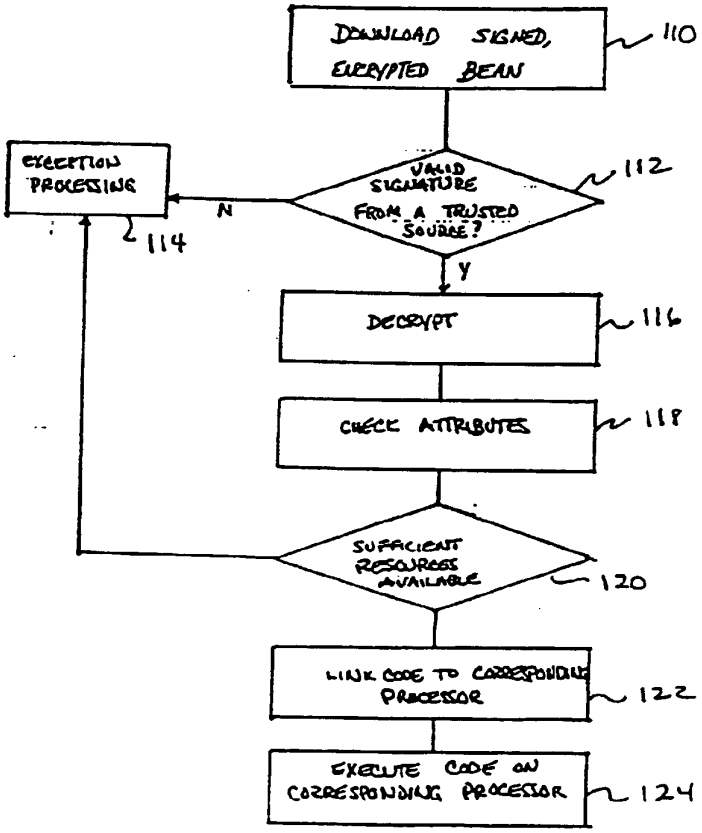


FIG. 6

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 98 31 0312

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
P,X	WO 98 40978 A (SAGEM ;DEMEURE JEAN ANDRE (FR); DIMECH JEAN MARC (FR)) 17 September 1998 * page 4, line 22 - line 27 * * page 5, line 25 - line 28 * * page 8, line 26 - line 29 *	1,2,4, 11,12, 18,19	H04Q7/32 H04B1/38 G06F9/38
P,X	EP 0 869 691 A (DEUTSCHE TELEKOM AG) 7 October 1998 * column 2, line 4 - line 22 *	1-4, 11-14, 18,19	
A	GB 2 310 575 A (WESTINGHOUSE ELECTRIC CORP) 27 August 1997 * page 5, line 16 - line 25 *	1,2,12, 18,19	
A	WO 97 26750 A (CELLPORT LABS INC) 24 July 1997 * page 18, line 6 - page 22, line 26 *	1,12,18, 19	
A	US 4 862 407 A (FETTE BRUCE A ET AL) 29 August 1989 * column 4, line 49 - line 58 * * column 13, line 14 - line 18 *	1,12,18, 19	TECHNICAL FIELDS SEARCHED (Int.Cl.6) H04Q H04M G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
BERLIN	31 May 1999	Leouffre, M

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 98 31 0312

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

31-05-1999

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9840978 A	17-09-1998	FR 2760917 A	18-09-1998
		FR 2760918 A	18-09-1998
		AU 6921998 A	29-09-1998
EP 0869691 A	07-10-1998	DE 19713965 A	08-10-1998
GB 2310575 A	27-08-1997	AU 1264397 A	28-08-1997
WO 9726750 A	24-07-1997	US 5732074 A	24-03-1998
		AU 1525197 A	11-08-1997
		CA 2243454 A	24-07-1997
		EP 0875111 A	04-11-1998
US 4862407 A	29-08-1989	NONE	

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 12/00	A2	(11) International Publication Number: WO 99/05600
		(43) International Publication Date: 4 February 1999 (04.02.99)

(21) International Application Number: PCT/US98/15340

(22) International Filing Date: 24 July 1998 (24.07.98)

(30) Priority Data:
08/901,776 28 July 1997 (28.07.97) US

(71) Applicant: APPLE COMPUTER, INC. [US/US]; Law Dept., M/S: 38-PAT, 1 Infinite Loop, Cupertino, CA 95014 (US).

(72) Inventors: GARST, Blaine; 3307 Bay Court, Belmont, CA 94002 (US). SERLET, Bertrand; 218 Colorado Avenue, Palo Alto, CA 94301 (US).

(74) Agents: HECKER, Gary, A. et al.; Hecker & Harriman, Suite 2300, 1925 Century Park East, Los Angeles, CA 90067 (US).

(81) Designated States: CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD AND APPARATUS FOR ENFORCING SOFTWARE LICENSES

(57) Abstract

The present invention comprises a method and apparatus for enforcing software licenses for resource libraries such as an application program interface (API), a toolkit, a framework, a runtime library, a dynamic link library (DLL), an applet (e.g. a Java or ActiveX applet), or any other reusable resource. The present invention allows the resource library to be selectively used only by authorized end user software programs. The present invention can be used to enforce a "per-program" licensing scheme for a resource library whereby the resource library is licensed only for use with particular software programs. In one embodiment, a license text string and a corresponding license key are embedded in a program that has been licensed to use a resource library. The license text string and the license key are supplied, for example, by a resource library vendor to a program developer who wants to use the resource library with an end user program being developed. The license text string includes information about the terms of the license under which the end user program is allowed to use the resource library. The license key is used to authenticate the license text string. The resource library in turn is provided with means for reading the license text string and the license key, and for determining, using the license key, whether the license text string is authentic and whether the license text string has been altered. Resource library functions are made available only to a program having an authentic and unaltered license text string.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND APPARATUS FOR ENFORCING SOFTWARE LICENSES

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION

The present invention relates generally to the distribution of computer software, and more particularly to a method and apparatus for automated enforcement of computer software licenses.

2. BACKGROUND ART

Some computer software programs use so-called "resource libraries" to provide part of their functionality. There is usually a license fee required to use a resource library. Under current schemes, it is not always possible to charge the license fee to all users of a resource library. This problem can be understood by comparing software structures that use resource libraries with basic software structures that do not.

Basic Software Structure

Figure 1 illustrates a basic software structure. In the example of Figure 1, the software comprises two layers. These layers are the operating system 110, and the application program 120. Operating system 110 is responsible for controlling the allocation and usage of hardware resources such as memory, central processing unit (CPU) time, disk space, and peripheral devices. Operating system 110 provides a variety of specific functions that can be

utilized by a variety of software programs such as application program 120. Application program 120 provides specific end user functions, such as word processing, database management, and others. Application program 120 communicates with the computer hardware via functions provided by operating system 110. Operating system 110 provides an interface between hardware 100 and application program 120.

Resource Libraries

Figure 2 illustrates a second software structure. The software structure of Figure 2 contains an additional layer of software, resource library 215, interposed between application program 220 and operating system 110. Resource library 215 provides a pre-packaged set of resources or routines that can be accessed by software programs such as application program 220 during execution. These resources provide higher level functions than those provided by operating system 210. For example, these resources may provide routines for managing a graphical user interface, for communicating with other computers via a network, or for passing messages between program objects. Typically, resource library 215 provides one or more resources or functions that can be used by many different software programs. By using the pre-packaged resources provided by resource library 215, a software program such as application program 220 can be made smaller and program development time can be shortened because the program itself need not include code to provide the functions provided by resource library 215.

In addition to application programs, resource libraries are used by other types of software programs, including device drivers, utility programs and other resource libraries.

Resource library 215 constitutes any set of one or more resources that exists separately from an application program or other software program and that can be used by more than one software program. For example, resource library 215 may comprise an application program interface (API), a toolkit, a framework, a resource library, a dynamic link library (DLL), an applet, or any other reusable resource, including an application program that can be accessed by another program (e.g. by using object linking and embedding (OLE)). Examples of resource libraries include Windows DLL's (DLL's used with the Microsoft Windows (TM) operating environment), the Apple Macintosh (TM) toolkit, the OpenStep API from NeXT Software, Inc., OLE enabled application programs such as Microsoft Word (TM), Java packages, and ActiveX applets.

A software program typically utilizes a resource provided by a resource library by sending an appropriate message to the resource library and supplying the parameters required for the resource to be executed. Assuming the appropriate parameters have been supplied, the resource executes, and an appropriate response message is returned to the requesting program.

A software program may use resources provided by several different resource libraries, a resource library may be used by several different programs, and a resource library may itself use other resource libraries. Figure 3 illustrates a computer system that includes several programs and several resource libraries. In the example of Figure 3, there are two application

programs 300 and 310, and three resource libraries 320, 330, and 340. Application program 300 uses resources provided by operating system 110 and by resource libraries 320 and 330. Application program 310 uses resources provided by operating system 110 and by resource libraries 330 and 340. The resources of resource library 330 are thus shared by application programs 300 and 310.

License Fee

Generally, computer software is licensed to an end user for a fee. The end user pays a single purchase price or license fee in exchange for the right to use the end user program on a computer system. Resource libraries are often packaged or "bundled" with an end user program by the maker of the program such that the end user receives a copy of resource libraries required by a program when the end user buys a copy of the program. The price of the resource library is built into the end user program price. The end user program developer, in turn, pays a royalty to the resource library vendor for the right to bundle and resell the resource library.

Since a resource library can be used with multiple end user programs, once the end user receives a copy of the resource library, the end user can use the resource library with any other program that is compatible with the resource library. In this case, the resource library vendor receives no additional revenue when the vendor's resource library is used with additional programs. Accordingly, it would be desirable for a resource library vendor to be able to ensure that an end user can use the resource library only with programs for which a license fee has been paid to the vendor for use of the

resource library. Thus there is a need for a software mechanism for enforcing software license agreements that automatically ensures that a resource library can only be used by programs that have been licensed for use with the resource library by the resource library vendor.

SUMMARY OF THE INVENTION

The present invention comprises a method and apparatus for enforcing software licenses for resource libraries. The term "resource library" as used herein refers to any reusable software resource that is usable by more than one program or other resource library. The term "resource library" includes, but is not limited to, an application program interface (API), a toolkit, a framework, a runtime library, a dynamic link library (DLL), an applet (e.g. a Java or ActiveX applet), an application program whose functionality can be accessed by other programs (e.g. using OLE) or any other reusable resource. The present invention allows the resource library to be selectively used only by authorized end user software programs. The present invention can be used to enforce a "per-program" licensing scheme for a resource library whereby the resource library is licensed only for use with particular software programs, as well as site licenses and other licensing schemes.

In one embodiment, an access authorization indicator such as a license text string and a corresponding license key are embedded in a program that has been licensed to use a resource library. The license text string and the license key are supplied, for example, by a resource library vendor to a program developer who wants to use the resource library with an end user program being developed.

The license text string includes information about the terms of the license under which the end user program is allowed to use the resource library. In one embodiment, the license key is an algorithmic derivation, such as, for example, a digital signature, of the license text string that is used to

authenticate the license text string. The resource library in turn is provided with a checking routine that includes means for reading the license text string and the license key, and for determining, using the license key, whether the license text string is authentic and whether the license text string has been altered. Resource library functions are made available only to a program having an authentic and unaltered license text string.

In one embodiment, the license key constitutes the resource library vendor's digital signature of the license text string. The resource library has a checking routing for verifying the resource library vendor's digital signature. The resource library is unlocked and made available for use with the requesting program only if the license text string is verified as authentic by the resource library. For a given program, only the resource library proprietor can produce a license key for a particular license agreement that will unlock the resource library for that program and that program only. Any modification of the license key or the license agreement text string in the requesting software program is detected by the checking routine, causing the resource library to remain locked. The license text string may also specify an expiration date for the license, in which case the resource library is unlocked only if the expiration date has not yet occurred.

In one embodiment, a per-site enforcement method is provided, in which any software program present at a given user site works with the resource library once the resource library is provided with the proper per-site license key.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates an example of a software structure.

Figure 2 illustrates an example of a software structure including a resource library.

Figure 3 illustrates an example of a software structure including several application programs and resource libraries.

Figure 4 illustrates an embodiment of a computer system that can be used with the present invention.

Figure 5 illustrates a software structure of one embodiment of the present invention.

Figure 6 illustrates a software structure of one embodiment of the present invention.

Figure 7 is a flow chart illustrating the operation of one embodiment of the present invention.

Figure 8 illustrates a software structure of one embodiment of the present invention.

Figure 9 illustrates a software structure of one embodiment of the present invention.

Figure 10 is a flow start illustrating the operation of one embodiment of the present invention.

Figure 11 is a flow start illustrating the operation of one embodiment of the present invention.

Figure 12 is a flow start illustrating the operation of one embodiment of the present invention.

Figure 13 illustrates a software structure of an embodiment of the present invention using the OpenStep API.

Figure 14 illustrates an embodiment of the invention in which the resource library is an applet.

DETAILED DESCRIPTION OF THE INVENTION

A method and apparatus for enforcing software licenses is described. In the following description, numerous specific details are set forth in order to provide a more thorough description of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known features have not been described in detail so as not to obscure the invention.

Computer System

The present invention can be implemented on any of a variety of computer systems, including, without limitation, network computers, special purpose computers, and general purpose computers such as the general purpose computer illustrated in Figure 4. The computer system shown in Figure 4 includes a CPU unit 400 that includes a central processor, main memory, peripheral interfaces, input-output devices, power supply, and associated circuitry and devices; a display device 410 which may be a cathode ray tube display, LCD display, gas-plasma display, or any other computer display; an input device 430, which may include a keyboard, mouse, digitizer, or other input device; non-volatile storage 420, which may include magnetic, re-writable optical, or other mass storage devices; a transportable media drive 425, which may include magnetic, re-writable optical, or other removable, transportable media, and a printer 450. The computer system may also include a network interface 440, which may include a modem, allowing the computer system to communicate with other systems over a communications network such as the Internet. Any of a variety of other configurations of

computer systems may also be used. In one embodiment, the computer system comprises an Intel Pentium (tm) CPU and runs the Microsoft Windows 95 (tm) operating environment. In another embodiment, the computer system comprises a Motorola 680X0 series CPU and runs the NeXTStep operating system.

When a computer system executes the processes and process flows described herein, it is a means for enforcing software licenses.

The invention can be implemented in computer program code in any desired computer programming language.

Licensing Module

Figure 5 is a block diagram illustrating software components of one embodiment of the present invention. As shown in Figure 5, this embodiment, like the prior art embodiment of Figure 2, includes computer hardware 100, operating system 110, application program 220 and resource library 215. However, the present invention adds two additional components: Program licensing module 500 and resource library licensing module 510. These modules are shown in greater detail in Figure 6.

Figure 6 illustrates program licensing module 500 and resource library licensing module 510 in one embodiment of the present invention. As shown in Figure 6, program licensing module 500 contains license text string 600 and license key 610. License text string 600 contains data specifying terms of the software license agreement under which the resource library vendor

has licensed the program containing program licensing module 510 to use the vendor's resource library. For example, license text string 600 may include the following text:

Table 1: Example License Text String

```
"(c) Copyright 1997.  Resource Library Vendor, Inc. Program A is
licensed to use Resource Library D.  No expiration date.  This
license may not be legally copied or transferred to another
program."
```

In the example shown in Table 1, license text string 600 specifies the name of the resource library vendor ("Resource Library Vendor, Inc.), the name of the program licensed to use the resource library ("Program A"), and the name of the resource library that has been licensed ("Resource Library D"). License text string 600 also indicates that the license has "No expiration date."

License key 610 is algorithmically derived from license text string 600. In one embodiment, license key 610 comprises a digital signature of the resource library vendor.

A digital signature is a mechanism that has been developed to help ensure the integrity of electronic messages. A digital signature is used to authenticate an electronic message and to determine whether an electronic message has been altered.

One form of digital signature uses a message digest. A message digest is a value that is generated when an electronic message is passed through a one way encryption process ("digesting process") such as a hashing routine. An ideal digesting process is one for which the probability that two different electronic messages will generate the same message digest is near zero. In this

form of digital signature, both the originator and the recipient need to know which digesting process is being used. The originator generates the electronic message, and generates a message digest by passing the electronic message through the digesting process. The originator digitally signs the resulting message digest, for example by performing an algorithmic operation on the message digest using the originator's private key. Alternatively, instead of generating a message digest and signing the message digest, a sender may sign the message itself.

To verify the authenticity of a digitally signed message, the recipient obtains the electronic message and the digital signature of the sender. The recipient verifies the digital signature using an appropriate verification process. For example, in one embodiment, the recipient verifies the digital signature by performing an algorithmic process on the digital signature using the sender's public key. The verification process verifies that the electronic message was (1) digitally signed by the sender, and (2) that the electronic message content was not changed from the time that it was signed to the time that the digital signature was verified.

In the present embodiment of the invention, the "message" that is digitally signed is license text string 600. The signer is the resource library vendor. The result is license key 610.

License text string 600 and license key 610 are used by resource library licensing module 510 to verify that a requesting program has been licensed to use the resource library. As shown in Figure 6, resource library licensing module 510 includes a license verification module 620. When a program

requests access to the resource library, resource library licensing module 510 reads license text string 600 and license key 610 from the requesting program. In one embodiment, license text string 600 and license key 610 are sent to the resource library by the requesting program along with a request for access to the resource library. In another embodiment, resource library licensing module 510 reads license text string 600 and license key 610 from a constant definition section of the requesting program.

Resource library licensing module 510 uses license key 610 to verify the content of license text string 600 in the same manner as a digital signature is used to verify an electronic message. Using license verification module 620, resource library licensing module 510 verifies that license text string 600 is authentic (i.e. was generated by the resource library vendor) and unaltered. If the verification process is unsuccessful, indicating that the digital signature is not good, resource library licensing module 510 refuses the requesting program's request for access to the resource library. If the verification process is successful, resource library licensing module 510 inspects the license to determine any license limitations included in license text string 600.

The example license text string 600 shown in Table 1 above identifies "Program A" as the program that is licensed to use the resource library, and states that the license has "No expiration date." Resource library licensing module 510 obtains the name of "Program A" from license text string 600, and checks whether the requesting program is Program A. If the requesting program is a program other than Program A, access to the resource library is denied.

Rather than specifying "No expiration date" as in the present example, license text string 600 may specify an expiration date and/or a beginning date for the license. If any such dates are specified in license text string 600, resource library licensing module 510 checks to make sure that the current date falls within the period of validity of the license prior to granting access to the resource library. If the current date is not within the license's period of validity, the requesting program is denied access to the resource library.

Access Procedure

The process used by a resource library to grant or deny access to a requesting program in one embodiment of the invention is illustrated in Figure 7. In one embodiment, this process occurs the first time a program requests access to a resource library. In another embodiment, this process occurs each time the resource library receives a request for access.

As shown in Figure 7, the process begins with a requesting program making a request to use the resource library at step 700. At step 705, the resource library obtains the requesting program's license text and license key. The license text and license key may, for example, be included in the request, or the resource library may read the license text and license key from a constant declaration area of the requesting program, or the resource library may obtain the license text and license key by some other means.

After obtaining the license text and license key, the resource library verifies the authenticity of the license text, using the license key, at step 710. At step 725, a the resource library determines whether the verification is

successful. If the authenticity of the license text is not verified, access to the resource library is denied at step 730.

If the verification of the authenticity of the license text is successful, the resource library checks the license terms included in the license text at step 735. At step 740, the resource library determines whether a limited validity period is specified in the license text. If no validity period is specified, the process continues on to step 755. If a validity period is specified, the resource library checks whether the validity period has expired at step 745. The validity period will have expired either if the current date is before a beginning date specified in the license text or if the current date is after an expiration date specified in the license text. If the validity period has expired, access to the resource library is denied at step 750.

If the validity period has not expired, processing continues to step 755. At step 755, the resource library determines whether the requesting program is the same program as the program specified in the license text. If the requesting program is not the program specified in the license text, access to the resource library is denied at step 760. If the requesting program is the program specified in the license text, the resource library checks whether there are any other license terms contained in the license text at step 765. If there are no other license terms, access to the resource library is granted at step 770. If there are other license terms, the resource library checks whether those terms are satisfied at step 775. If the terms are not satisfied, access to the resource library is denied at step 780. If the terms are satisfied, access to the resource library is granted at step 785.

The invention may be implemented in the Objective-C language. Objective-C is essentially the ANSI C language with object messaging extensions. A full description of the Objective-C language appears in "Object-Oriented Programming and the Objective-C Language," published by Addison-Wesley (ISBN 0-201-63251-9) (1993), and incorporated by reference herein. However, the invention can also be implemented in any other suitable computer programming language.

As described below, the invention can be implemented by embedding appropriate segments of program code in the source code of a program that uses a resource library and in the source code of the resource library itself. The resource library is compiled to produce an executable implementation which can be linked to a compiled and executable version of the program.

Application Program Interface (API)

In one embodiment of the invention, the resource library is an application program interface ("API"). An API has three major functions: it receives requests from an application program to carry out fundamental operations such as receiving user input or displaying output; it converts each request into a form understandable by the particular operating system then in use; and it receives responses and results from the operating system, formats them in a uniform way, and returns them to the application program.

APIs generally are prepared in an executable implementation which is compiled specifically for the underlying operating system. This is necessary because different operating systems provide different calling mechanisms and

communications methods for such primitive operations as reading and writing a mass storage device. For example, an API may provide a "draw(x,y)" function that can be called by an application program to draw a point at coordinates (x,y) on the display device of a computer system. Upon receipt of a draw(x,y) request from an application program, the API converts the request into a command or function call specific to the operating system then in use. For example, the API might convert the draw(x,y) request into a series of machine instructions to load registers with the x,y values and call an operating system function or generate an interrupt. The person writing the application program need not worry about such details.

In some cases the API refers to or calls functions located in an external function library such as a set of device drivers rather than directly calling the operating system. Device drivers are small executable programs that enable the operating system to address and work with particular hardware devices such as video adapters and printers. Device drivers also constitute a form of resource library.

Depending on the operating system, the API can be prepared in any of several executable formats such as a runtime library, device linked library (DLL), or other executable file. The API is provided to the end user in one of these object code versions, or "implementations," of the API. In industry usage the term API can refer to a definition or specification of functions in the API, to the source code of the API that implements such functions, or to the executable version of such source code which is ultimately distributed to and used by end users. Examples of APIs are the OpenStep API, available from

NeXT Software, Inc., Redwood City, California, and the Visual Basic DLL available from Microsoft Corporation, Redmond, Washington.

The term API as used herein also includes the Java programming language. Rather than being distributed in executable form, Java programs are distributed as packages of "bytecodes." The bytecodes are compiled at runtime into executable code by a Java Virtual Machine (JVM) resident on the computer on which the Java program is run. Different JVM's are used for different computer processors and operating systems. However, all JVM's read the same bytecode. Accordingly, Java bytecode programs and packages are platform independent. Java bytecode programs and packages need only be written in one form. The JVM's take care of adapting the bytecode to different computer platforms. Packages of Java bytecode can be used by different Java programs, and, as such, constitute resource libraries.

Generally the end user can buy the executable version of the API implementation separately from any particular application program from its creator or vendor, or the end user may buy the API implementation bundled with an application program that requires and uses the API to run.

In either case, the API implementation is installed in executable form in the end user's computer system (typically by copying it to a mass storage device such as a hard disk). After the API implementation is installed, the end user can launch (begin running) an application program which uses the API implementation. The application program locates the API implementation on the hard disk and references, calls, or is linked to the API implementation. In operation, when the application program needs to carry

out an operation implemented in the API implementation, such as drawing a line on the screen, the application program calls the appropriate function in the API implementation. The appropriate function in turn tells the operating system (or the device independent windowing extensions, or another device driver) how to execute the desired operation.

A significant advantage of the use of APIs is that an application program, such as a word processor, can be written to communicate only with the API, and not with the operating system. Such an application program can be moved or ported to a different operating system without modifying the program source code. Because of this, application programs written for APIs are said to be operating system independent, meaning that the application program source code can be moved without modification to another computer system having a different operating system, and recompiled and linked with an API implementation prepared for that operating system. The ability to move unmodified application source code to different operating systems is a key advantage of using APIs.

However, from the point of view of API vendors, APIs also have the significant disadvantage that an end user needs only one copy of the API to run multiple application programs which are compatible with the API. Since the API provides generic input, output, and processing functions, it will work with a variety of different end user application programs. Some software vendors desire to restrict use of their API implementations to one application, or to require the end user to purchase a key to the API for each application acquired by the end user, so that the end user pays a different or larger fee to use additional application programs.

The present invention provides a way to arrange a resource library such as an API to work only with particular authorized application or other end user programs.

API License Embodiment

As is well known in the art, the source code of a computer program can be divided into several components including a variables declaration area, a constant declaration area, and a procedure definition area. Figure 9 illustrates an embodiment of the present invention that is used with an API. As shown in Figure 9, in this embodiment, an application program 900 is provided with a LicenseKeyString constant 902 and a LicenseAgreementString constant 904 in the constant declarations area 901 of the application program's source code. In the embodiment of Figure 9, LicenseKeyString 902 and LicenseAgreementString 904 are declared as global string constants.

In one embodiment, LicenseAgreementString 904 contains a text string, prepared by the vendor of the API, that describes in human readable text the license restrictions concerning use of the API applicable to the application program. For example, the LicenseAgreementString may read, "This API is licensed for individual internal use only for concurrent use only with Word Processor application program." The specific text of the LicenseAgreementString is prepared by the licensor of the API. The text can be any arbitrary combination of words, symbols, or numbers.

The LicenseKeyString 904 contains a key corresponding to and based upon the LicenseAgreementString 902. For example, the LicenseKeyString can be a digital signature of the LicenseAgreementString prepared by providing the LicenseAgreementString and a private key of the API vendor to a digital signature process. The precise method of generating the LicenseKeyString is not critical, provided that only the licensor of the API can generate a unique LicenseKeyString corresponding to the LicenseAgreementString. The values of the two strings are created by the vendor of the API and are provided to the person or company that is developing the end user application program (for example, the API vendor can send the two string values to the application program developer by e-mail). The application program developer is instructed by the API vendor to place the string declarations in the source code of the developer's end user application program. The two values may be public, so the API vendor or developer need not keep the values secret or hidden from users of the end user application program. The two strings are compiled into the executable form (or, in the case of Java, the bytecode packages) of the application program. This binds the LicenseKeyString and LicenseAgreementString into the executable code (or bytecode) of the application program.

As further shown in Figure 9, API 920 is provided with an UNLOCK function 923 and a CHECK LICENSE function 921 for testing whether the LicenseKeyString matches the LicenseAgreementString. In the embodiment of Figure 9, the CHECK LICENSE function 921 includes sub-function CHECK 922.

API Procedure

Figure 10 is a flow diagram of processing steps of the UNLOCK function 923. The process of Figure 10 may, for example, be carried out at runtime, when both the application program and the API are compiled, linked, and running.

The UNLOCK function is called by the API upon initialization of the API, for example, upon being called by application program 900 or by some other calling function, object, or program (the "calling entity"). Processing begins at step 1002. The UNLOCK function first checks to see whether the API has been provided with a site license that allows the API to be used with any calling entity on the computer in which the API has been installed. In this embodiment, a site license is indicated by adding an appropriate LicenseKeyString and LicenseAgreementString to the API when the API is installed. This process is described in greater detail below. An appropriate LicenseAgreementString may, for example, be "API site license granted. This API may be used with any application program at the site at which it is installed." The corresponding appropriate LicenseKeyString may, for example, be derived by applying the API vendor's private key and a digital signature process to the LicenseAgreementString.

The process of checking for a site license begins at step 1004 where the UNLOCK function locates and extracts (to the extent they have been provided to the API) a LicenseKeyString and a LicenseAgreementString from within the API. Control is then passed to step 1006 where the function tests whether the API is licensed under a site license for unrestricted use with any application

program. The test of step 1006 is accomplished by verifying the authenticity of the LicenseKeyString and LicenseAgreementString extracted from the API, and, if authentic, determining whether the LicenseAgreementString indicates that a site license has been granted.

The authenticity of the LicenseAgreementString and LicenseKeyString is determined by passing the LicenseAgreementString, the LicenseKeyString, and a copy of the API vendor's public key stored in the API implementation to the CHECK process 922. CHECK process 922 uses a digital signature authentication ("DSA") process to verify the authenticity of the LicenseAgreementString.

The DSA process used by CHECK process 922 can be any digital signature authentication process capable of reading an input string and a key purportedly representing the digital signature of the input string, applying an appropriate authentication process, and determining the validity of the input string by testing whether the key constitutes the signatory's digital signature of the input string. An exemplary DSA process is disclosed, for example, in U.S. Patent Application Serial No. 08/484,264, "Method and Apparatus for Digital Signature Authentication," assigned to the assignee hereof. The DSA technology of RSA Data Security, Inc. also can be adapted for use with the invention. A per-session cache can be used to improve execution speed of the CHECK process.

If the LicenseKeyString is determined to be the API vendor's valid digital signature of the LicenseAgreementString, the LicenseAgreementString is inspected to determine whether it indicates that a site license has been

granted. If the LicenseAgreementString does so indicate, the test of step 1006 succeeds and control is passed to step 1014. At this point the UNLOCK function returns a positive result to the calling entity, and allows the calling entity to use the API.

If the test of step 1006 fails, control is passed to step 1008 where the UNLOCK function extracts and reads the LicenseKeyString and LicenseAgreementString from a data segment (for example, the compiled constant declarations area) of the calling entity. Alternatively, the calling entity may transmit the LicenseKeyString and the LicenseAgreementString to the API. Having obtained the calling entity's LicenseKeyString and LicenseAgreementString, control is passed to step 1010 where the function tests whether the calling entity is licensed to use the API. This test comprises two parts. One part, using CHECK process 922 as described above, determines whether the LicenseAgreementString is a LicenseAgreementString validly issued by the API vendor. A second part examines the LicenseAgreementString for the terms of the included license, and determines whether those terms are met. If the result is positive then control is passed to step 1014. At this point, use of the API with the calling entity is authorized and the API returns control to the calling entity so that the calling entity resumes normal execution.

If the result is negative then the calling entity is not licensed to use the API, and control is passed to step 1012. At step 1012 the API generates an error message such as "API Not Licensed For Use With This Application program," and declines access to the calling entity.

Steps 1006 and 1010 carry out the license tests by calling the CHECK LICENSE function 921 shown in Figure 9 and Figure 11. Processing steps of the CHECK LICENSE function 921 are illustrated in Figure 11.

The process flow of the CHECK LICENSE function starts at step 1102. Control is passed to step 1104 where the CHECK LICENSE function assembles the LicenseKeyString 902, LicenseAgreementString 904, and a copy of the API vendor's public key 1106 as function call arguments, in preparation for calling the CHECK function 922. As discussed more fully below, the public key 1106 is prepared by the API vendor based upon a secret private key. The three arguments are passed to the CHECK function at step 1108.

If the CHECK function (described in greater detail below) returns a FAIL or false state, control is passed to step 1124 and the CHECK LICENSE function itself returns a fail state. If the CHECK function returns a PASS or true state, control is passed to step 1112 where the CHECK LICENSE function checks the terms of the license specified in the LicenseAgreementString. At step 1114, the CHECK LICENSE function checks whether the name of the calling entity is the same as the name of the licensed entity specified in the LicenseAgreementString. If the name of the calling entity is incorrect, control passes to step 1124, where a fail message is passed to the UNLOCK function.

If the name of the calling entity is correct, the CHECK LICENSE function tests whether the LicenseAgreementString contains an expiration date at step 1116. An expiration date can be placed in the LicenseAgreementString by the API vendor to establish a termination date after which use of the API by the calling entity is no longer allowed. CHECK

LICENSE may, for example, test for an expiration date by searching for a text string that indicates an expiration date, such as, for example, "expiration date" or "valid until."

If the test of step 1116 is positive, control is passed to step 1118 where the CHECK LICENSE function tests whether the current date, as maintained, for example by a computer clock or operating system, is greater than the expiration date found in the LicenseAgreementString. If the test of step 1118 passes, control is passed to step 1120. If the test of step 1118 fails, then CHECK LICENSE returns a FAIL message at block 1124.

At step 1120, the CHECK LICENSE function checks whether the LicenseAgreementString specifies any additional license terms. If there are no other terms, CHECK LICENSE returns a PASS message at block 1126. If there are other terms, CHECK LICENSE determines whether those terms are met at block 1122. If any of the other terms are not met, CHECK LICENSE returns a FAIL message at block 1124. If all of the additional terms are met, CHECK LICENSE returns a PASS message at block 1126.

The operation of the CHECK function called by CHECK LICENSE at block 1108 is illustrated in Figure 12. As shown in Figure 12, the purpose of the CHECK function is to verify the authenticity of a license agreement string by verifying that a corresponding license key string constitutes a valid digital signature of the license agreement string. The CHECK function begins at step 1202 and receives as input a LicenseKeyString, a LicenseAgreementString, and a vendor's public key in step 1203. The public key is generated by the resource library vendor using any known public/private key pair generation process, as

is well known in the field of cryptography. For example, key generation using Fast Elliptical Encryption (FEE) can be done, or Diffie-Hellman key generation can be used.

In step 1204 the CHECK function verifies that the LicenseKeyString comprises the digital signature of the LicenseAgreementString. In step 1208, the CHECK function tests whether the verification of step 1204 successfully verified the LicenseKeyString as comprising the digital signature of the LicenseAgreementString. If so, the LicenseAgreementString is valid, and CHECK returns a Boolean true or pass value. If not, the LicenseAgreementString is invalid, and CHECK returns false or failure.

Since the LicenseKeyString of the present embodiment comprises the digital signature of the LicenseAgreementString, the LicenseAgreementString cannot be changed in any way without the change being detected. Stated more generally, because the identifier (e.g. the LicenseKeyString) of the invention is a unique key mathematically derived from a particular text string that specifies license terms for a particular end user program (e.g. the LicenseAgreementString), the identifier can be used to detect any changes to the license terms. This prevents unauthorized modification of the text string from extending use of a resource library to an unlicensed program. For example, if an end user attempts to modify the expiration date using a debugger or machine language editor, the identifier will no longer match the license text string. Without knowing the private key of the vendor, the end user cannot generate a matching identifier.

When a 127-bit private key's is used by the vendor to create the identifier used in the present invention, a determined hacker attempting to forge the private key would need to exhaustively search the 127-bit space, requiring extensive computing resources and an impractical amount of time. Thus, the protection provided by the present invention cannot easily be cracked and the security of the invention as a whole is extremely high.

In addition to allowing per program resource library licensing, if the API vendor or licensor desires to grant a site license for the API to the end user, so that the API is licensed for use with any number of application programs, the API may be provided with a LicenseKeyString and a LicenseAgreementString providing for such unrestricted use. In this embodiment, the API vendor provides a site license key string to the end user as authorization to use the API with any number of applications and other end user programs at that site. The site license key string comprises a digital signature of a site license agreement string created by the API vendor. The site license agreement string may be pre-embedded in the API by the vendor. During installation of the API, an installation program provided with the API asks the end user whether a site license key is known. If so, the end user enters the site license key, and the installation program writes the site license key to a reserved location in the API. Thereafter, when the API initializes, the API tests for the presence of the site license key. If it is present, and it comprises a valid digital signature for the site license text string stored elsewhere in the API, the API is permitted to be used with any application program which is calling it.

OpenStep API

In one embodiment of the invention, the API used is the object-oriented OpenStep API 820 shown in Figure 8. A specification of the OpenStep API has been published by NeXT Software, Inc. under the title "OPENSTEP SPECIFICATION," dated October 18, 1994. Implementations of the OpenStep API include implementations for the Windows NT and Solaris operating systems that are available from NeXT Software, Inc. and SunSoft, Inc., respectively.

As shown in Figure 8, the OpenStep API 820 comprises computer program code organized as an Application Kit 802, Foundation Kit 808, and Display Postscript™ system 804. (Display Postscript™ is a trademark of Adobe Systems Incorporated.)

Application Kit 802 provides basic resources for interactive application programs that use windows, draw on the screen, and respond to user actions on the keyboard and mouse. Application Kit 802 contains components that define the user interface. These components include classes, protocols, C language functions, constants and data types that are designed to be used by virtually every application running under the OpenStep API. A principal purpose of Application Kit 802 is to provide a framework for implementing a graphical, event-driven application.

Foundation Kit 808 provides fundamental software functions or building blocks that application programs use to manage data and resources. Foundation Kit 808 defines basic utility classes and facilities for handling

multi-byte character sets, object persistency and distribution, and provides an interface to common operating system facilities. Foundation Kit 808 thus provides a level of operating system independence, enhancing the developer's ability to port an application program from one operating system to another.

Display Postscript system 804 provides a device-independent imaging model for displaying documents on a computer screen. Display Postscript is defined by Adobe Systems Incorporated. Display Postscript system 804 provides an application-independent interface to Postscript.

Separate from the API 820, but also logically located between the application program 800 and the operating system 810, is a set of device dependent windowing extensions 806. Extensions 806 enable Display Postscript system 804 to communicate with specific graphics and display hardware in the end user's computer system, such as the video memory or other video display hardware.

Figure 13 illustrates an embodiment of the invention used with the OpenStep API of Figure 8. As shown in Figure 13, in this embodiment, the license text string and the license key string of the invention are implemented in a property list area 1302 (Info.plist) of the application program code 800. Two string properties are added to the property list area 1302: NSLicenseAgreement 1304, that stores the software license terms applicable to application program 800, and NSLicenseKey 1306, that stores the license key corresponding to NSLicenseAgreement 1304. In this embodiment, as in the embodiment of Figure 9, NSLicenseKey 1306 is derived from the

NSLicenseAgreement string 1304 generated from the license agreement string using a digital signature process and a vendor's private key.

Example values of the two strings placed in the Info.plist are shown in Table 2.

Table 2 -- Info.plist Strings

```
NSLicenseKey = "Ab76LY2GbbO0GqK2KY17BqHy35";

NSLicenseAgreement = "(c) Copyright 1996, EOF AddOnTools
Inc., ReportWriter licensing agreement: This is
demonstration software valid until November 2, 1996.
This software cannot be legally copied.";
```

In the OpenStep embodiment of Figure 13, the UNLOCK function 1308 is implemented as part of Application Kit 802. In one embodiment, UNLOCK function 1308 is implemented by adding appropriate code to a non-redefinable private Application Kit function (such as, for example, _NXAppZone() in NSApplication.m). An example of source code that may be added is shown in Table 3.

Table 3 -- UNLOCK Code added in OpenStep API Implementation

```
static BOOL licenseChecked = NO;
if (! licenseChecked)
     {
     NSDictionary *info;
     NSString *key, *agreement;
     /* First check the unlimited (per-site) license */
     info = [NSDictionary
dictionaryWithContentsOfFile:@"/OpenStep/AppKit.dll/Info
.plist"]; // real path TBD
     key = [info objectForKey:@"NSLicenseKey"];
     agreement = [info
objectForKey:@"NSLicenseAgreement"];
     if (!NSCheckLicense(key, agreement))
          {
          /* now check for the per-app license */
          info = [[NSBundle mainBundle] infoDictionary];
```

```
            key = [info objectForKey:@"NSLicenseKey"];
            agreement = [info
      objectForKey:@"NSLicenseAgreement"];
            if (!NSCheckLicense(key, agreement))
                  {
                  NSLog(@"*** Sorry no valid license for
      %@", [NSApp appName]);
                  }
            }
      licenseChecked = YES;
      }
```

The NSCheckLicense() function, which is called twice in the code segment of Table 3, as shown in Figure 13, is implemented in the Foundation Kit portion 808 of the OpenStep API 820. The NSCheckLicense function 1310 corresponds to the CHECK LICENSE function 921 illustrated in Figure 9. The NSCheckLicense function 1310 verifies NSLicenseAgreement string 1304 using NSLicenseKey string 1306 and a digital signature authentication process. The NSCheckLicense function 1310 has the following definition:

```
extern BOOL NSCheckLicense(NSString *licenseKey,
      NSString *licenseAgreement);
```

The NSCheckLicense function 1310, like the Check License function 921 of Figure 9, applies a CHECK function 1312 to NSLicenseAgreement string 1304 and NSLicenseKey 1306, using the API vendor's public key, to determine the validity of NSLicenseAgreement string 1304. In the embodiment of Figure 13, CHECK function 1312 includes in its code a copy of the API vendor's public key 1314.

In the embodiment of Figure 13, API 820 includes a "GEN" process 1316 that can be used by an API vendor to rapidly generate license key strings for use by CHECK function 1312. GEN process 1316 receives as input a license agreement string and a secret private key, and produces as output a licensing

key string, using a digital signature generating process. The private key may, for example, be a 127-bit private key, although any other size private key may be used. The signature generating process used by GEN process 1316 is compatible with the digital signature authentication process used by CHECK function 1312. GEN process 1316 itself can be made entirely public and implemented in the API provided that the private key of the API vendor is maintained in secrecy. For example, the GEN process can be part of the OpenStep API Foundation Kit 808 as shown in Figure 13. GEN also can be maintained in a separate program module.

The logical relationship between GEN and CHECK is:

CHECK(GEN(LicenseAgreementString, PrivateKey), Public Key, LicenseAgreementString) => YES

CHECK(random1, random2) => NO with a very high probability

In one embodiment of the invention, a shell is provided for the GEN process. The shell can receive as input a license agreement template string, such as:

```
(c) Copyright 1995, %@, %@ licensing agreement; Demo
software valid until %@; This agreement cannot be
legally copied
```

where %@ represents additional data to be provided by the API vendor. The shell then asks the user (i.e. the API vendor) to input the additional data, for example a company name, a product name, an expiration date, from which the shell generates a specific license agreement string. The shell then asks for the private key and applies GEN to create a corresponding license key.

The same shell can be used for per-program license keys or per-site license keys, using different templates.

In one embodiment of the invention, an installer program is provided for installing a resource library on an end user computer. The installer program is provided with a feature enabling the end user to provide a site license key during installation. For example, if the resource library is the OpenStep API, additional code is added to the OpenStep API installer program. The user is asked during the installation of the resource library if the user has obtained a per-site license. If the user replies yes, the user is asked to enter the site license key string. In one embodiment, the user is also asked to enter the site license agreement string. In another embodiment, the site license agreement string is stored in the resource library, such as, for example, in the OpenStep API DLL Application Kit's Info.plist resource file. The site license key and site license agreement are validated by the CHECK LICENSE function as described above. Use of the resource library is permitted only if the site license key string input by the user corresponds to (i.e. is found to comprise the resource library vendor's digital signature of) the site license agreement string.

Java

The present invention may be used with resource libraries such as Java class files, Java applets, and Java bytecode packages. Figure 14 illustrates an embodiment of the invention in which the resource library is a Java applet. In the embodiment shown in Figure 14, an applet is called from an HTML page 1402 via applet tag 1404. Applet tag 1404 includes the name of the

applet's class file and applet parameters 1406. Applet parameters 1406 include a license agreement string parameter 1408 and a license key string parameter 1410. License agreement string parameter 1408 specifies a license agreement string that contains terms of a license to use the called for applet. License key string parameter 1410 specifies a license key used to authenticate the license agreement string. As in other embodiments of the invention, in this embodiment, the license key string comprises a digital signature by the resource library (applet) vendor of the license agreement string. Table 4 illustrates an example of applet tag 1404.

Table 4

```
<APPLET CODE="Applet.class"  WIDTH=250 HEIGHT=75>
<PARAM NAME=LicenseAgreementString VALUE="Web page
orderform.html licensed to use applet 'Applet.class'>
<PARAM NAME=LicenseKeyString VALUE="4kd094kak2rtx0kzq">
</APPLET>
```

In the example of Table 4, the license agreement string specifies the name of the HTML page ("orderform.html") and the name of the licensed applet ("applet.class").

As shown in Figure 14, applet 1434 is accessed when HTML page 1402 is loaded by a HTML browser 1430 running in a client computer 1420. In the embodiment of Figure 14, HTML browser 1430 runs on top of an API 1424 which in turn runs on top of operating system 1422. HTML browser 1430 includes a Java virtual machine 1432 for running Java applets.

Upon encountering applet tag 1404 while loading HTML page 1402, HTML browser 1430 retrieves the class files that constitute applet 1434 from storage locations on client computer 1420 and/or from one or more server

computers, as applicable. One of the class files includes CheckLicense class file 1436. After HTML browser 1430 has retrieved all the required components of applet 1434, applet 1434 is initialized. During initialization, or at a later time, the CheckLicense function provided by CheckLicense class file 1436 is called. As in other embodiments of the invention, the CheckLicense function determines whether the requesting entity (HTML page 1402) possesses a valid license to use the requested resource (applet 1434) by testing the authenticity of the license specified by LicenseAgreementString parameter 1408 using the license key specified by LicenseKeyString parameter 1410 and the applet vendor's public key 1438. If the CheckLicense function determines that HTML page 1402 possesses a valid license, applet 1434 is allowed to execute. If not, execution of applet 1434 is terminated, and an error message is sent to HTML browser 1430.

Thus, an improved method and apparatus for enforcing software licenses has been presented. Although the present invention has been described with respect to certain example embodiments, it will be apparent to those skilled in the art that the present invention is not limited to these specific embodiments. For example, although the invention has been described for use in stand-alone computer systems, the invention can be used to enforce licenses in a network environment as well. Further, although the operation of certain embodiments has been described in detail using specific software programs and certain detailed process steps, different software may be used, and some of the steps may be omitted or other similar steps may be substituted, without departing from the scope of the invention. Other embodiments incorporating the inventive features of the present invention will be apparent to those skilled in the art.

CLAIMS

1. In a computer operating environment comprising a software program and a software resource, an apparatus for limiting use of said software resource comprising:

an access authorization indicator associated with said software program;

means in said software resource for reading said access authorization indicator;

means in said software resource for determining whether said access authorization indicator is valid;

means for allowing access by said software program to said software resource only if said access authorization indicator is determined to be valid.

2. The apparatus of claim 1 wherein said access authorization indicator comprises terms of a license for use of said software resource.

3. The apparatus of claim 1 wherein said access authorization indicator comprises terms of a site license.

4. The apparatus of claim 1 wherein said access authorization indicator is embedded in said software program.

5. The apparatus of claim 1 wherein said software resource comprises an API.

6. The apparatus of claim 1 wherein said software resource comprises a runtime library.

7. The apparatus of claim 1 wherein said software resource comprises a dynamic link library.

8. The apparatus of claim 1 wherein said software resource comprises an applet.

9. The apparatus of claim 1 wherein said software resource comprises a bytecode package.

10. The apparatus of claim 1 wherein said software resource comprises an OLE enabled application program.

11. The apparatus of claim 4 wherein said access authorization indicator is specified in a constant declaration area of said software program.

12. The apparatus of claim 4 wherein said access authorization indicator comprises a property of a property list of said software program.

13. The apparatus of claim 1 further comprising an identifier associated with said access authorization indicator and wherein said means for determining the validity of said access authorization indicator comprises means for determining whether said access authorization indicator is valid based on said identifier.

14. The apparatus of claim 13 further comprising means for receiving said identifier from an end user.

15. The apparatus of claim 14 further comprising means for storing said identifier in said software resource.

16. The apparatus of claim 13 wherein said identifier is embedded in said software program.

17. The apparatus of claim 13 wherein said identifier comprises a digital signature of said access authorization indicator.

18. The apparatus of claim 16 wherein said identifier is specified in a constant declaration area of said software program.

19. The apparatus of claim 16 wherein said identifier comprises a property of a property list of said software program.

20. The apparatus of claim 17 wherein said means for determining whether said access authorization indicator is valid based upon said identifier comprises a means for digital signature authentication.

21. The apparatus of claim 2 further comprising means for determining whether said terms of said license are met.

22. The apparatus of claim 13 wherein:

said software program comprises said access authorization indicator and said identifier;

said access authorization indicator comprises terms of a license for use of said software resource;

said identifier comprises a digital signature of said access authorization indicator.

23. In a computer operating environment, a method for limiting use of a software resource comprising:

receiving a request from a software program to use said resource;

obtaining an access authorization indicator associated with said software program;

determining whether said access authorization indicator is valid;

allowing said software program to use said software resource only if said access authorization indicator is determined to be valid.

24. The method of claim 23 wherein said access authorization indicator comprises terms of a license for use of said software resource.

25. The method of claim 24 wherein said license comprises a site license.

26. The method of claim 23 wherein said access authorization indicator is embedded in said software program.

27. The method of claim 23 wherein said software resource comprises an API.

28. The method of claim 23 wherein said software resource comprises a runtime library.

29. The method of claim 23 wherein said software resource comprises a dynamic link library.

30. The method of claim 23 wherein said software resource comprises an applet.

31. The method of claim 23 wherein said software resource comprises a bytecode package.

32. The method of claim 23 wherein said software resource comprises an OLE enabled application program.

33. The method of claim 26 wherein said access authorization indicator is specified in a constant declaration area of said software program.

34. The method of claim 26 wherein said access authorization indicator comprises a property of a property list area of said software program.

35. The method of claim 23 wherein said determining the validity of said access authorization indicator comprises determining whether said access authorization indicator is valid based on an identifier associated with said access authorization indicator.

36. The method of claim 35 further comprising accepting said identifier from a user.

37. The method of claim 36 further comprising storing said identifier in said software resource.

38. The method of claim 35 wherein said identifier is embedded in said software program.

39. The method of claim 35 wherein said identifier comprises a digital signature of said access authorization indicator.

40. The method of claim 38 wherein said identifier is specified in a constant declaration area of said software program.

41. The method of claim 38 wherein said identifier comprises a property of a property list area of said software program.

42. The method of claim 35 wherein a digital signature authentication means is used in determining whether said access authorization indicator is valid based upon said identifier.

43. The method of claim 24 further comprising determining whether said terms of said license are met.

44. The method of claim 35 wherein:

said software program comprises said access authorization indicator and said identifier;

said access authorization indicator comprises terms of a license for use of said software resource;

said identifier comprises a digital signature of said access authorization indicator.

45. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform a method for limiting use of a software resource, said method comprising:

receiving a request from a software program to use said resource;

obtaining an access authorization indicator associated with said software program;

determining whether said access authorization indicator is valid;

allowing said software program to use said software resource only if said access authorization indicator is determined to be valid.

46. The program storage device of claim 45 wherein said access authorization indicator comprises terms of a license for use of said software resource.

47. The program storage device of claim 46 wherein said license comprises a site license.

48. The program storage device of claim 45 wherein said access authorization indicator is embedded in said software program.

49. The program storage device of claim 45 wherein said software resource comprises an API.

50. The program storage device of claim 45 wherein said software resource comprises a runtime library.

51. The program storage device of claim 45 wherein said software resource comprises a dynamic link library.

52. The program storage device of claim 45 wherein said software resource comprises an applet.

53. The program storage device of claim 45 wherein said software resource comprises a bytecode package.

54. The program storage device of claim 45 wherein said software resource comprises an OLE enabled application program.

55. The method of claim 48 wherein said access authorization indicator is specified in a constant declaration area of said software program.

56. The program storage device of claim 48 wherein said access authorization indicator comprises a property of a property list area of said software program.

57. The program storage device of claim 45 wherein said determining the validity of said access authorization indicator comprises determining whether said access authorization indicator is valid based on an identifier associated with said access authorization indicator.

58. The program storage device of claim 57 wherein said method further comprises accepting said identifier from a user.

59. The program storage device of claim 58 wherein said method further comprises storing said identifier in said software resource.

60. The program storage device of claim 57 wherein said identifier is embedded in said software program.

61. The program storage device of claim 57 wherein said identifier comprises a digital signature of said access authorization indicator.

62. The program storage device of claim 60 wherein said identifier is specified in a constant declaration area of said software program.

63. The program storage device of claim 60 wherein said identifier comprises a property of a property list area of said software program.

64. The program storage device of claim 57 wherein a digital signature authentication means is used in determining whether said access authorization indicator is valid based upon said identifier.

65. The program storage device of claim 46 in which said method further comprises determining whether said terms of said license are met.

66. The program storage device of claim 57 wherein:

said software program comprises said access authorization indicator and said identifier;

said access authorization indicator comprises terms of a license for use of said software resource;

said identifier comprises a digital signature of said access authorization indicator.

67. An article of manufacture comprising:

a computer readable medium having computer readable program code embodied therein for accessing a resource library, said computer readable program code in said article of manufacture comprising:

computer readable program code embodying an access authorization indicator for accessing said resource library.

68. The article of manufacture of claim 67 wherein said access authorization indicator comprises terms of a license for use of said software resource.

69. The article of manufacture of claim 67 wherein said computer readable program code comprises a software program and wherein said access authorization indicator is embedded in said software program.

70. The article of manufacture of claim 67 wherein said software resource comprises an API.

71. The article of manufacture of claim 67 wherein said software resource comprises a runtime library.

72. The article of manufacture of claim 67 wherein said software resource comprises a dynamic link library.

73. The article of manufacture of claim 67 wherein said software resource comprises an applet.

74. The article of manufacture of claim 67 wherein said software resource comprises a bytecode package.

75. The article of manufacture of claim 67 wherein said software resource comprises an OLE enabled application program.

76. The article of manufacture of claim 69 wherein said access authorization indicator is specified in a constant declaration area of said software program.

77. The article of manufacture of claim 69 wherein said access authorization indicator comprises a property of a property list of said software program.

78. The article of manufacture of claim 67 further comprising computer readable program code embodying an identifier associated with said access authorization indicator.

79. The article of manufacture of claim 78 wherein said identifier is embedded in said software program.

80. The article of manufacture of claim 78 wherein said identifier comprises a digital signature of said access authorization indicator.

81. The article of manufacture of claim 78 wherein said identifier is specified in a constant declaration area of said software program.

82. The article of manufacture of claim 78 wherein said identifier comprises a property of a property list of said software program.

83. The article of manufacture of claim 78 wherein:

said software program comprises said access authorization indicator and said identifier;

said access authorization indicator comprises terms of a license for use of said software resource;

said identifier comprises a digital signature of said access authorization indicator.

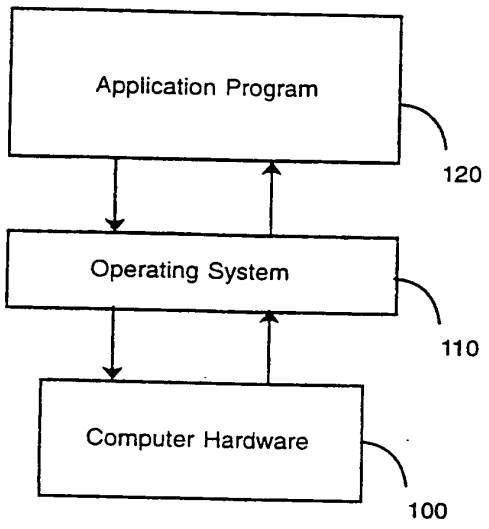


FIG. 1

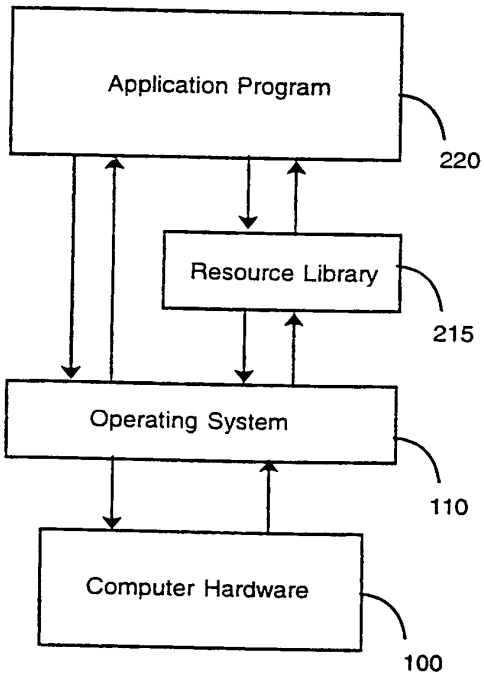


FIG. 2

FIG. 3

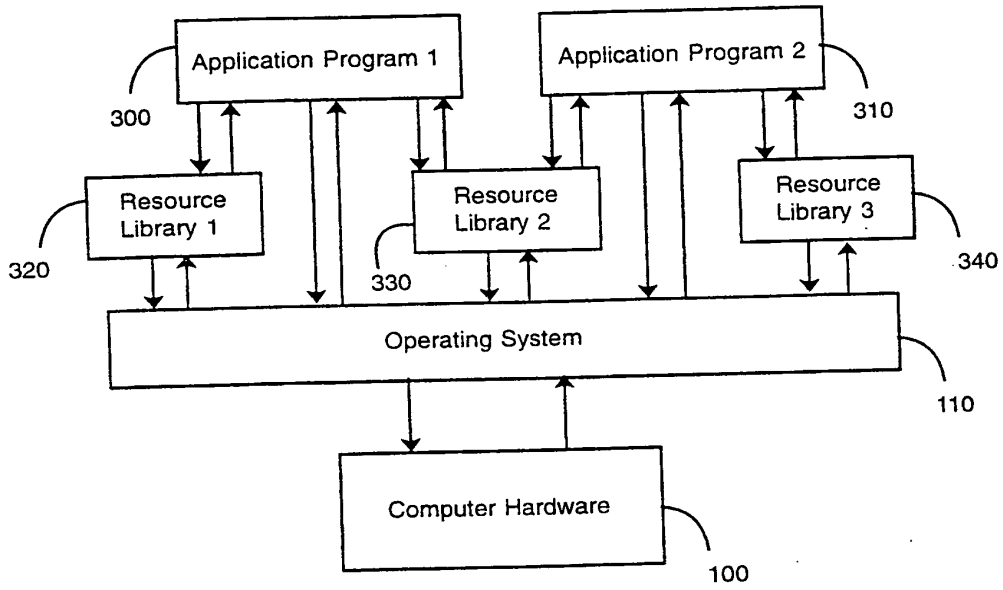
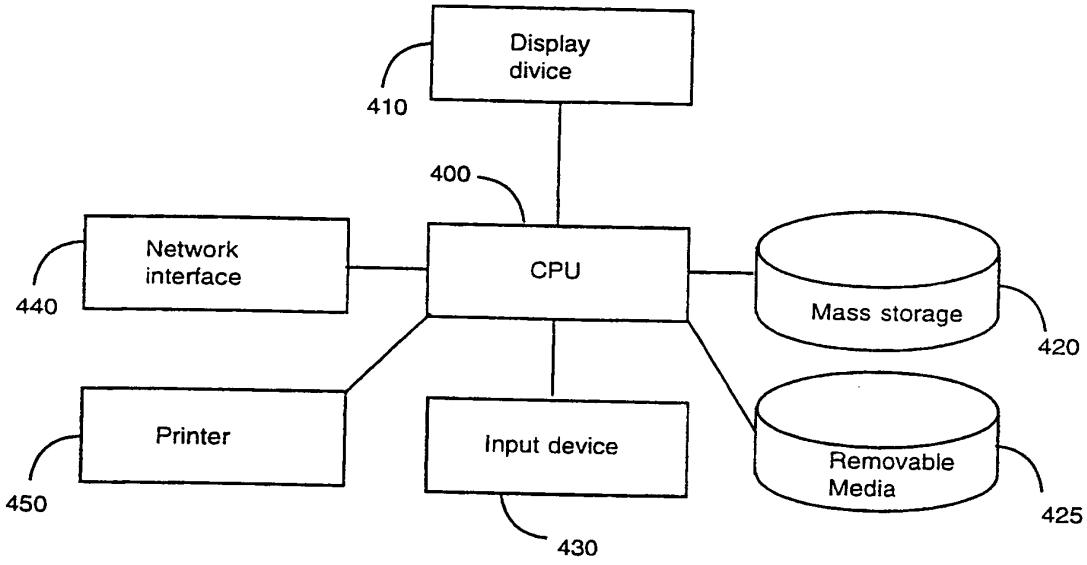


FIG. 4



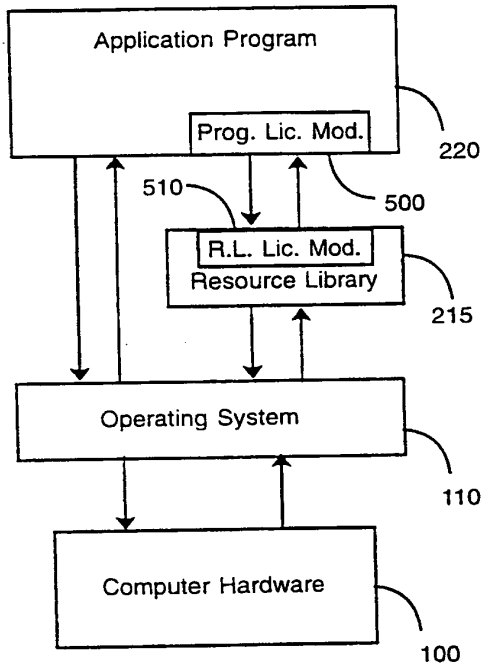


FIG. 5

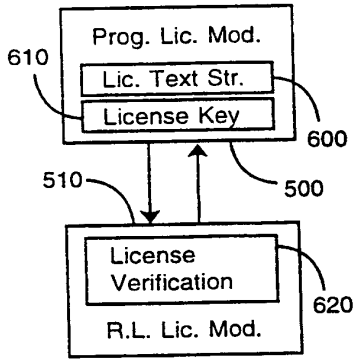


FIG. 6

FIG. 7

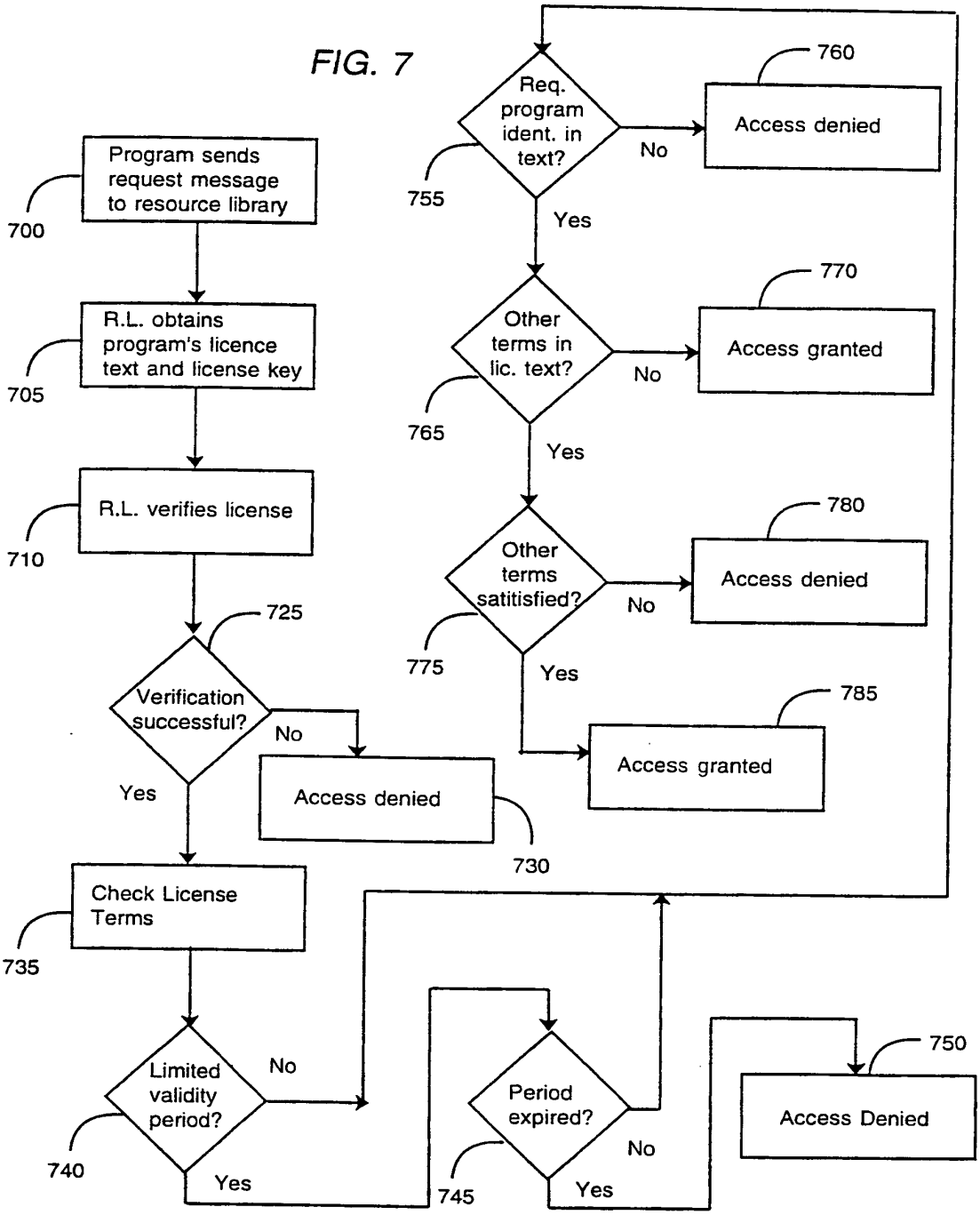


FIG. 8

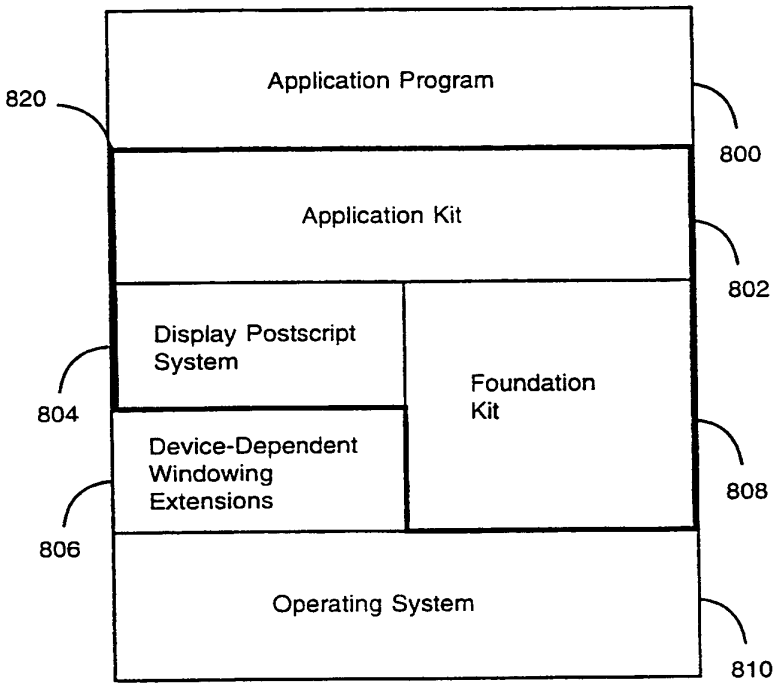
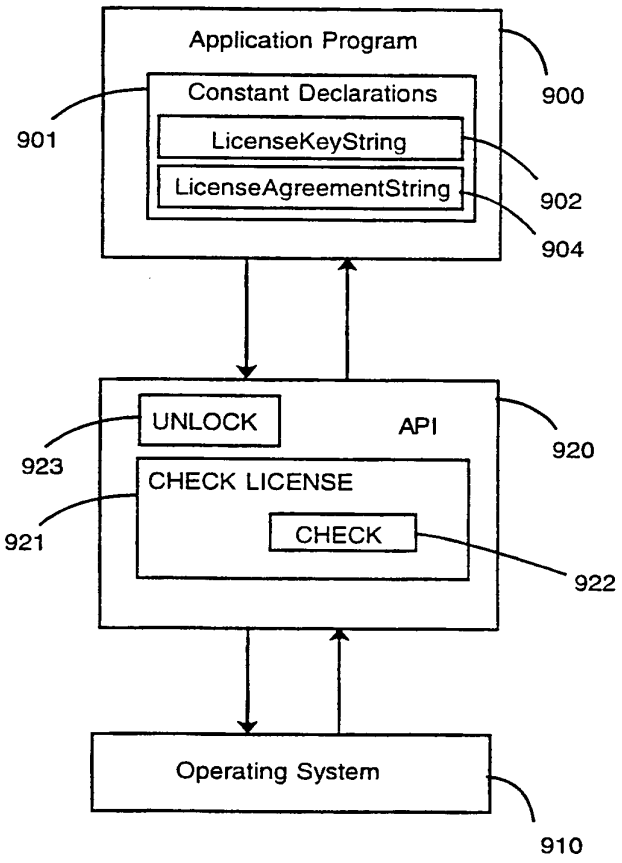


FIG. 9



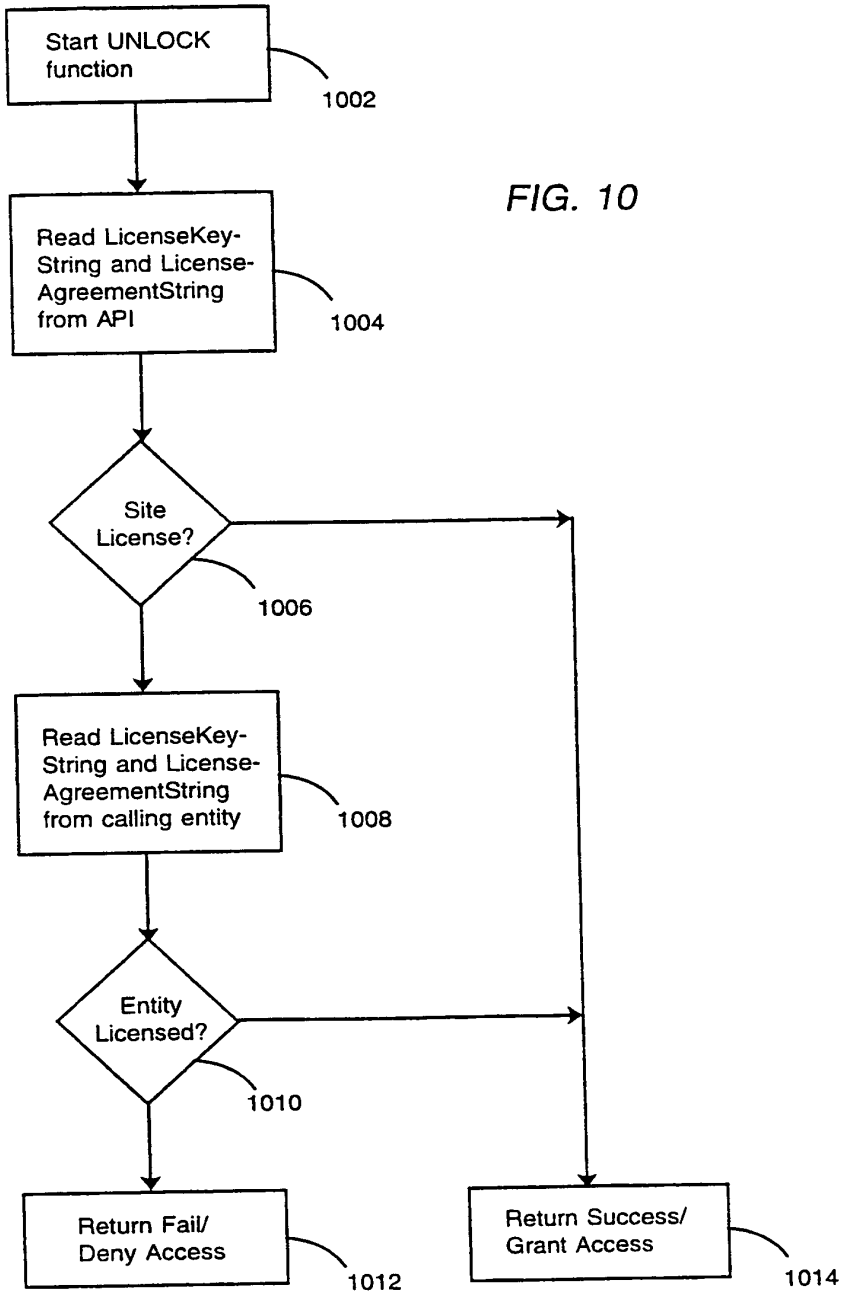


FIG. 10

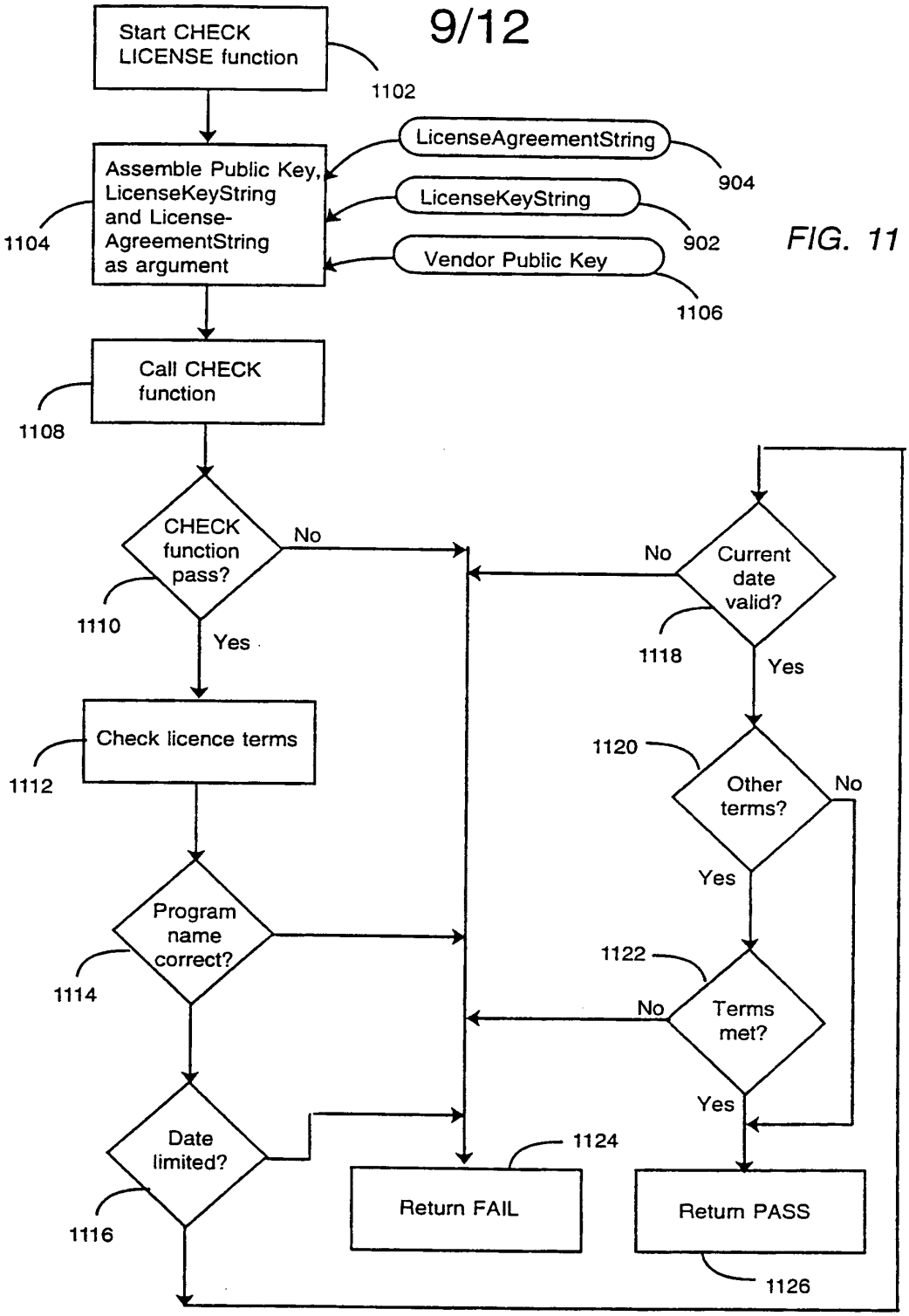


FIG. 11

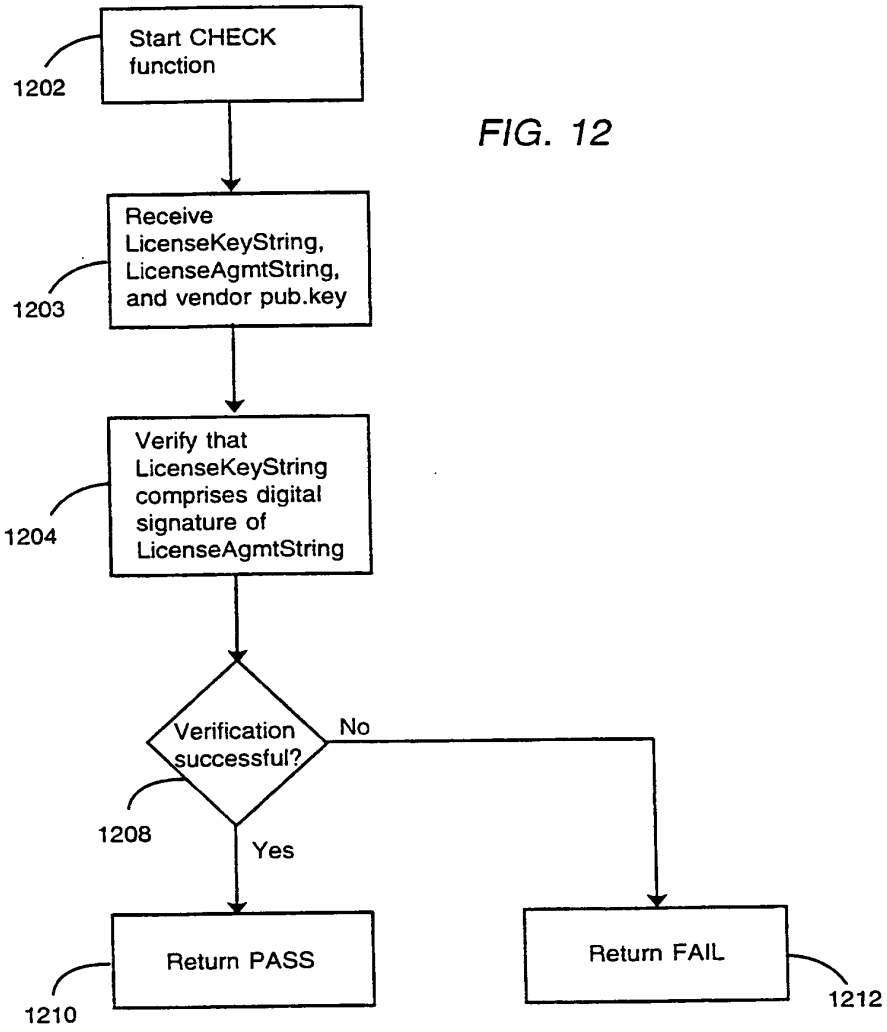


FIG. 12

FIG. 13

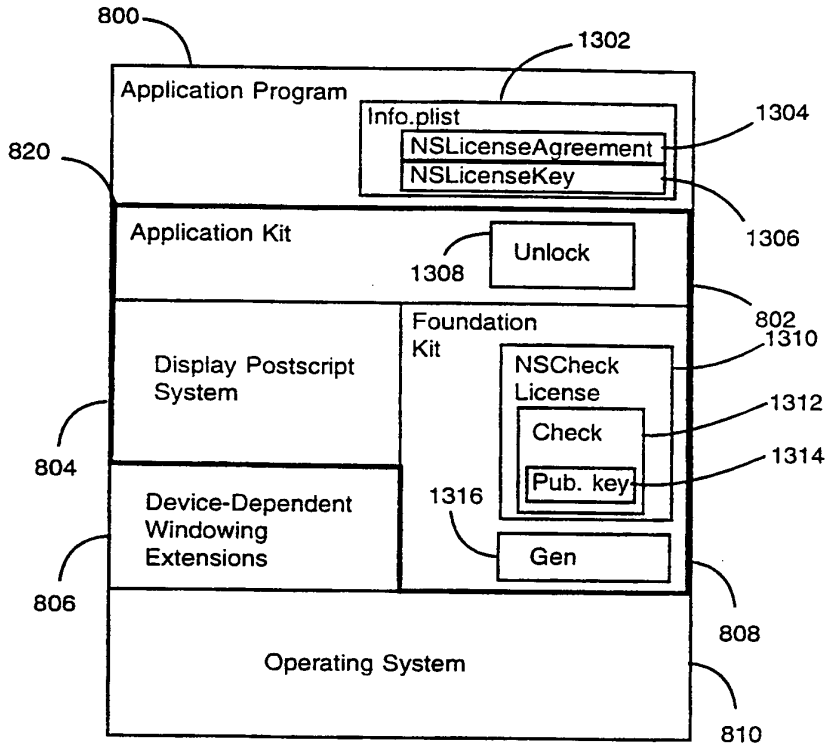
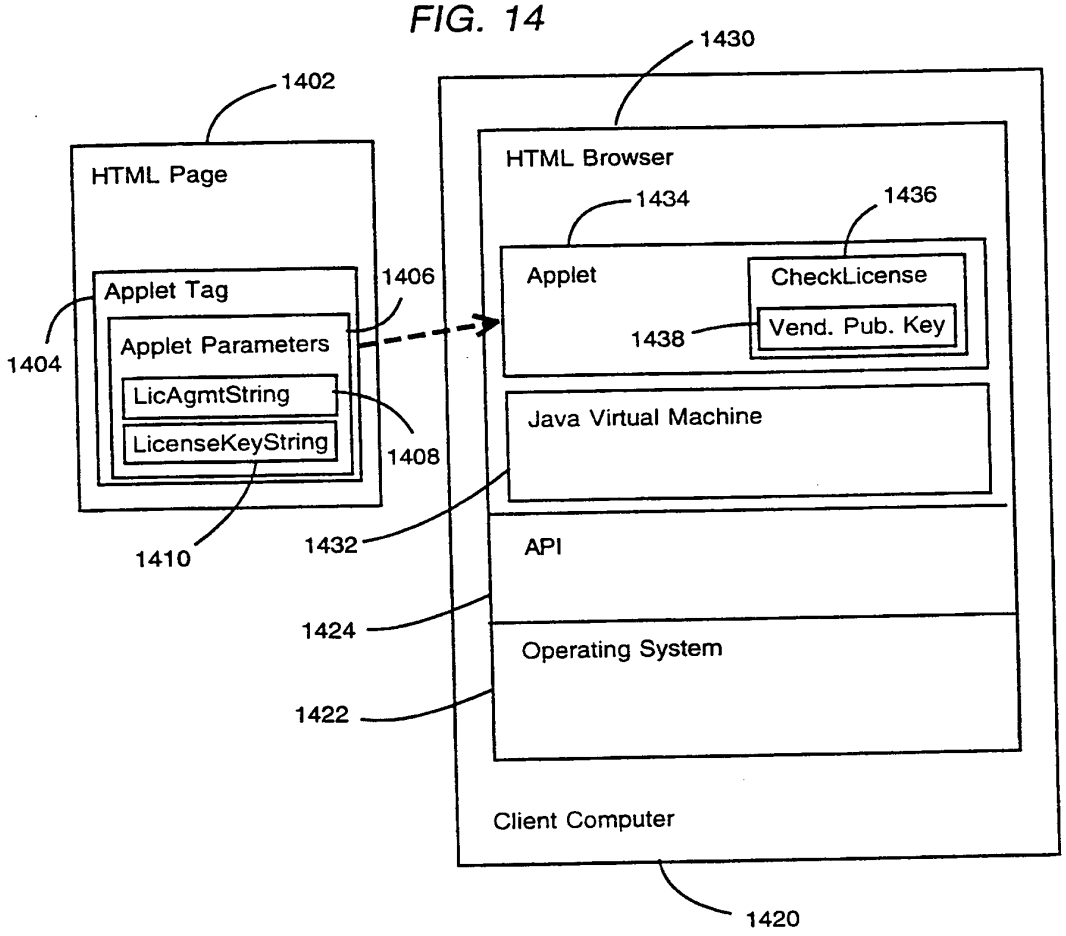


FIG. 14



PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 1/00, 9/46

A3

(11) International Publication Number: WO 99/05600

(43) International Publication Date: 4 February 1999 (04.02.99)

(21) International Application Number: PCT/US98/15340

(22) International Filing Date: 24 July 1998 (24.07.98)

(30) Priority Data:
08/901,776 28 July 1997 (28.07.97) US

(71) Applicant: APPLE COMPUTER, INC. [US/US]; Law Dept., M/S: 38-PAT, 1 Infinite Loop, Cupertino, CA 95014 (US).

(72) Inventors: GARST, Blaine; 3307 Bay Court, Belmont, CA 94002 (US). SERLET, Bertrand; 218 Colorado Avenue, Palo Alto, CA 94301 (US).

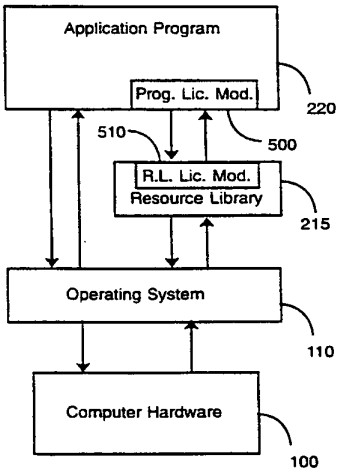
(74) Agents: HECKER, Gary, A. et al.; Hecker & Harriman, Suite 2300, 1925 Century Park East, Los Angeles, CA 90067 (US).

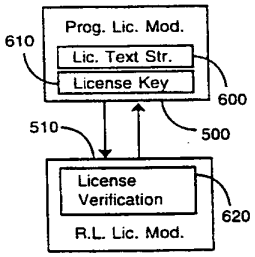
(81) Designated States: CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(88) Date of publication of the international search report:
14 May 1999 (14.05.99)

(54) Title: METHOD AND APPARATUS FOR ENFORCING SOFTWARE LICENSES





(57) Abstract

The present invention comprises a method and apparatus for enforcing software licenses for resource libraries such as an application program interface (API), a toolkit, a framework, a runtime library, a dynamic link library (DLL), an applet (e.g. a Java or ActiveX applet), or any other reusable resource. The present invention allows the resource library to be selectively used only by authorized end user software programs. The present invention can be used to enforce a "per-program" licensing scheme for a resource library whereby the resource library is licensed only for use with particular software programs. In one embodiment, a license text string and a corresponding license key are embedded in a program that has been licensed to use a resource library. The license text string and the license key are supplied, for example, by a resource library vendor to a program developer who wants to use the resource library with an end user program being developed. The license text string includes information about the terms of the license under which the end user program is allowed to use the resource library. The license key is used to authenticate the license text string. The resource library in turn is provided with means for reading the license text string and the license key, and for determining, using the license key, whether the license text string is authentic and whether the license text string has been altered. Resource library functions are made available only to a program having an authentic and unaltered license text string.

BNSDOCID: <WO___9905600A3_I_>

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

INTERNATIONAL SEARCH REPORT

Inte[redacted]al Application No
PCT/US 98/15340

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F1/00 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	EP 0 667 572 A (IBM) 16 August 1995 see abstract; figure 4 see page 4, line 53 - page 5, line 27 see claims 1-9	1-83
Y	EP 0 570 123 A (FISCHER ADDISON M) 18 November 1993 see abstract; figure 3D see claims 1-57	1-83
A	WO 97 14087 A (ERICKSON JOHN S) 17 April 1997 see abstract; figures 1,4,10,12 see page 7, paragraph 2 - page 9, paragraph 1	1-83
A	EP 0 778 512 A (SUN MICROSYSTEMS INC) 11 June 1997	

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
12 March 1999	19/03/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Powell, D

Form PCT/ISA/210 (second sheet) (July 1992)

BNSDOCID: <WO___9905600A3_I_>

INTERNATIONAL SEARCH REPORT

[redacted]formation on patent family members

[redacted]onal Application No
[redacted]T/US 98/15340

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0667572 A	16-08-1995	JP 7230380 A	29-08-1995
		US 5673315 A	30-09-1997
EP 0570123 A	18-11-1993	US 5412717 A	02-05-1995
		AU 3820993 A	18-11-1993
		CA 2095087 A	16-11-1993
		JP 6103058 A	15-04-1994
		US 5311591 A	10-05-1994
WO 9714087 A	17-04-1997	US 5765152 A	09-06-1998
		AU 7662496 A	30-04-1997
EP 0778512 A	11-06-1997	US 5708709 A	13-01-1998
		JP 9288575 A	04-11-1997

Form PCT/ISA/210 (patent family annex) (July 1992)

PTO/SB/06 (07-06)
Approved for use through 1/31/2007. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

PATENT APPLICATION FEE DETERMINATION RECORD

Substitute for Form PTO-875

Application or Docket Number	Filing Date	☐ To be Mailed
10/381,219	03/20/2003	

APPLICATION AS FILED – PART I

	(Column 1)	(Column 2)	SMALL ENTITY ☐		OR	OTHER THAN SMALL ENTITY	
FOR	NUMBER FILED	NUMBER EXTRA	RATE ($)	FEE ($)		RATE ($)	FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X $ =		OR	X $ =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X $ =			X $ =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).						
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))							
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL			TOTAL	

APPLICATION AS AMENDED – PART II

AMENDMENT		(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	SMALL ENTITY RATE ($)	SMALL ENTITY ADDITIONAL FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =		OR	X $ =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =		OR	X $ =	
	☐ Application Size Fee (37 CFR 1.16(s))									
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

AMENDMENT		(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)		RATE ($)	ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =		OR	X $ =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =		OR	X $ =	
	☐ Application Size Fee (37 CFR 1.16(s))									
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.

** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".

*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".

The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Legal Instrument Examiner:
/LINDA HUMES/