

-continued

Term	General Definition
Package	A collection of components that is signed and packaged for distribution
Manifest (package, build and device)	A file which describes the contents of a package. Note that in the build environment, there is a package manifest file (with a .BIB extension) that contains entries which describe the names of each file in the package; and a different manifest file, a device side manifest (with a .DSM extension), is a binary file that describes the complete set of information about a package (described below with reference to FIG. 9). As described with reference to FIG. 3, a build manifest file is created from the package manifest file, and used by a package generation process that generates the device manifest file (and the rest of the package).
Shadow Order Tool	Build tool which processes the Component Relationships File and generates the Package Shadow Files.

[0058]

Exten.	File Type	General Description
.bib	Binary image builder	Contains a list of files which should be included in the resultant image
.reg	registry file	file contains a list of registry (setting) information to be included in the image
.pkd.xml	Package Definition File	XML file in the build tree which defines what packages exist (package name, guid, version, loc buddy)
.cpm.csv	Component to Package Mapping file	CSV file in the build tree which maps MODULES tags and files to packages
.crf.csv	Component Relationships File	Text file in the build tree which defines the relationships (shadow and dependency) between MODULES tags
.bsm.xml	Build Side Manifest File	an XML file translation of the .bib (binary image builder) file
.psf	Package Shadow File	Intermediate file in the build environment (one per package, as in <packagename>.psf) which lists the packages that need to be shadowed by the named package.
.dsm	Device Side Manifest file	File on the device (one per package) which describes the package (files in the package, name, guid, signature, version, shadows, dependencies, CRC, root certs, etc.)
.pkg.cab	Canonical Package file	Full version of a package which contains the complete file for all files in the package.
.pku.cab	Update Package file	A file which facilitates the update of a device from a specific version of a single package to a different version of the same package. This file may contain binary diffs of individual files, or complete files, whichever minimizes the size of the update package.
.pks.cab	Super Package file	A file which contains a collection of Update Packages and/or Canonical Packages.

[0059] Turning to FIG. 2 of the drawings, as part of the overall package generation process, during the build process a build manifest file 202 is created by taking a binary image builder file 204 (the .bib file, also referred to as a package manifest file) for the operating system image and a compo-

nent to package mapping file 206 as inputs. As described above, the build manifest file 202 specifies the file contents for a particular package.

[0060] FIG. 3 generally shows an example build manifest file creation process 208 by which the build manifest file 202 is created from the binary image builder file 204. As can be seen from FIG. 3, in general, each line in the binary image builder file 204 is parsed; lines that parse as a .bib file entry having valid tags (looked up in an execute-in-place table (steps 310 and 312) may be compressed as desired (step 318), and written to the build manifest file 202 (step 320).

[0061] The resultant build manifest file 202 is an XML file that is validated with the following XSD:

```

<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://tempuri.org/ManifestCollection.xsd"
elementFormDefault="qualified"
xmlns="http://tempuri.org/ManifestCollection.xsd"
xmlns:mstns="http://tempuri.org/ManifestCollection.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ManifestCollection">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="BibFileEntry" minOccurs="1"
maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="DeviceName"
type="xs:string"
minOccurs="1" maxOccurs="1" />
              <xs:element name="ReleaseName"
type="xs:string"
minOccurs="1" maxOccurs="1" />
              <xs:element name="Region"
type="xs:string"
minOccurs="1" maxOccurs="1" />
              <xs:element name="Section"
type="xs:string"
minOccurs="1" maxOccurs="1" />
              <xs:element name="Attribs"
type="xs:string"
minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

[0062] Similar to the way in which the build manifest file 202 was created, the registry for the operating system image is broken up (into what is sometimes referred to as an RGU file) and assigned to packages based on a similar algorithm. An example of such a process 210 (FIG. 2) is depicted in FIG. 4, wherein the registry settings 212 (FIG. 2) associated with valid tags (step 414), including their registry key names as appropriate, are written into the package RGU file (step 422). Note that the registry key names are written to that package's RGU file whenever a package being processed changes (steps 418 and 420).

[0063] Further, XML files 218 (FIG. 2, e.g., containing other settings) may be broken up and assigned to specific packages. An example process 220 for doing this is depicted in FIG. 5, wherein valid tags for the children nodes (evaluated at step 508) are assigned to a node of the package (step 514).