

are self-satisfying. In other words, selecting one feature will also cause other features, on which the selected feature is dependent, to be selected. The build system thus ensures that a self-consistent monolithic operating system image results from any random selection of features on the part of the user. However, as described above, a monolithic operating system image has disadvantages relative to an image comprised in which subparts of the image can be individually updated.

[0036] In accordance with an aspect of the present invention, images including software updates are built from self-contained, secure entities. A fundamental update primitive is referred to as a package, wherein in general, a package is an encapsulation of a set of files that are versioned the same and are updated as a unit. The present invention provides a package format as one which is self-describing, a significant improvement when updating images, and one that facilitates replacing only a component part of an image.

[0037] Images are built from packages and contain both executable code and data that may be applied to storage. Note that executable code is customized to the virtual address space environment of the embedded device at install time; for example, depending on the underlying flash memory technology, some devices allow executable code to be run directly from the flash (execute-in-place, which means that the code cannot be stored in a compressed format) while other devices require that the code be copied (including decompressing the code as necessary) into RAM in order to run. In keeping with an aspect of the present invention, image update technology uses packages to break the operating system image into updateable components that can be updated in isolation, while maintaining any cross-component dependencies.

[0038] In accordance with an aspect of the present invention, there is provided a system and method that maps operating system features (comprising files, metadata, configuration information and so forth) into packages as part of a software build process. The packages can be used for initial device installation, and also for updates. As described below, software update packages may be in various forms, for example some may contain only the changes (deltas) to a previous update, while others may contain files that entirely replace other files. One other type of package can contain other packages.

[0039] The package concept described herein is a component part of the updating process. The process by which operating system features (generally an abstract concept that maps to specific files, metadata, configuration information, and so forth) are mapped into packages provides an ease-of-use advantage for the user (e.g., the provider of the image), including that instead of having to identify the

lowest level components of an operating system image, the user is able to refer to higher level handles that describe a full set of associated files, metadata, configuration information and so forth for a particular aspect of the operating system image. Note that as used herein, the terms “feature” and “component” ordinarily may be used interchangeably. By referring to a feature handle, the user is able to gain an advantage in terms of managing packaging complexity through abstraction. For example, rather than specifically mapping the executable module, dynamically-linked libraries (DLLs), resource/data files, registry information, and the like for browsing component software (e.g., Internet Explorer) and individually mapping each part into a package, the present invention enables the user to refer to this associated information with a single “Internet Explorer” handle and thus map it at the feature level to a package.

[0040] The present invention also provides packaging logic to handle cases where specific files/settings are shared between related features, which the user in turn chooses to map to differing packages. The logic in various algorithms (described below) generally ensures that individual files/settings are mapped to the correct package, given a number of possible higher level package mapping requests. In situations in which the logic cannot determine a correct course of action, messaging is provided to the user to indicate any issues that require user intervention to resolve.

[0041] Further, packages optionally convey dependency information. For example, when the contents of a package rely on the contents of another package, the relationship is captured during the build process and encoded into the package for later analysis during the installation process. The present invention also provides a mechanism (via feature-level dependency specifications) by which packages acquire the dependency information.

[0042] In accordance with an aspect of the present invention, feature selections can be mapped to an array (one or more) of packages, resulting in specific files and settings being mapped to an appropriate package. To accomplish this correctly, logic resolves conflicts and dependencies below the feature level. For example, if two features logically refer to the same specific file and the features are mapped to different packages, the logic determines into which package the shared file is to be placed.

[0043] In one implementation, package files are defined at build time by three different files: a package definition file (pkd), a component mapping file (cpm), and a component relations file (crf). The pkd file defines the global attributes of the contents of a package. The pkd file is an XML file that is validated during the build process against the following XSD:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://tempuri.org/Packages.xsd" elementFormDefault="qualified"
  xmlns="http://tempuri.org/Packages.xsd" xmlns:mstns="http://tempuri.org/Packages.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="PackageCollection">
    <xs:complexType>
      <xs:sequence>
```