Problem Statement

J2ME or Java ME is ripe with device fragmentation. Add to that the limited memory available for midlet suites, it is imperative that developer tools provide developers with the help necessary to develop midlets for a wide range of devices. A sampling of the areas in which fragmentation occurs:

- Available API's
- Screen size and color depth
- Maximum allowable jar size (64k up to 200k)
- Maximum runtime memory (200k up to multiple megabytes)
- Processor speed

In addition, localization generally needs to be performed at build time to avoid the memory cost of having multiple copies of the localizable resources for each language.

Necessary Tools

In order to support developers, there needs to be a number of tools available which may be involved in the ultimate solution.

Device Management

Device management in this case goes beyond the simple "Installed JREs" functionality currently available in the Eclipse JDT. For one thing, it is much more common for a developer to have a large number of devices defined than "Installed JREs". Each device brings with it a set of libraries, available security domains and other device information. In addition, some devices such as the generic devices provided by the Sun Wireless Toolkit, provide a set of optional profile libraries that may be enabled and disabled for that device. Device management needs to include the ability to import devices, remove defined devices and create derived devices that include optional libraries.

Code Management

In many cases, it becomes necessary to fill in the blanks between two disparate devices. For example, building a skeleton implementation of the MIDP 2.0 CustomItem class for use on a MIDP 1.0 device.

Due to the disparity between the devices, there will often be a need for one-off implementations for each project. This type of disparity can be managed using a combination of conditional compilation/preprocessing of source files as well as allowing for alternative source folders which may be switched based on the device.

Over time, an organization will most likely also collect a library of functionality which may be applied and reapplied to each new project. These libraries may be managed by allowing for alternative prerequisite projects which may be switched based on the selected device.

In summary, code management should include the following functionality:

- Multiple prerequisite project references, selectable based on device.
- Multiple source folders, switchable based on device.
- Conditional compilation/preprocessing.

Resource Management

Resource management is necessary in order to manage localizable text as well as resources such as images. The ability to have switchable prerequisite projects and source folders should provide the necessary support to appropriately manage resources.

Configuration Management

The ability to group all of above tools into a coherent “configuration” is an absolute necessity. Managing devices, prerequisite projects, alternative source folders and conditional compilation settings one by one would be painful at best. What is needed is the ability to group together all of these settings into a higher level configuration that may be applied to a project. Each configuration should provide the ability to control the following items:

- Default device
- Default device/emulator parameters (for instance to configure maximum memory)
- Associated source folders
- Associated prerequisite projects
- Conditional compilation symbol definitions

Build Support

In addition to the needs for management of device fragmentation, many organizations will have build requirements that extend beyond those currently provided by the development environment.

In many cases, a mobile application is backed by a J2EE server application. In these cases, it is necessary to allow the build of the mobile application to be integrated with the build of the server application. Ant build scripts are the logical choice for allowing the mobile application build to be integrated into another build.

Each project is likely to be associated with a number of individual configuration definitions. Building a project must allow for the ability to build a series of configurations in a single batch.

User Scenarios

Add Devices

- User opens Device Management section of workbench preferences
- User chooses to add new devices.
- User is prompted for a root file system directory in which to search.
- System recursively searches through file system directories searching for known devices.
    - (Plug-ins to framework must be given the chance to identify and “own” devices)
- User is prompted with a list of found devices. All devices found are initially selected.
- User deselects devices that they are not interested in having available.
- User presses OK to leave dialog and devices are added to system.

Create Mobile Project

- Similar to previous project creation use case, except a device is selected as the basis for the project.

Select Project Device

- User opens project properties.
- User selects available device for the project.
- User applies changes.
- Project is rebuilt based on new settings.

Remove Devices

- User opens Device Management
- User selects all devices to remove
- User selects “Delete” key or “Remove” button
- User is prompted to confirm removal
- Devices are removed
  - What happens to projects associated with this device?

Create New Configuration

- User opens Configuration Management interface in workbench preferences
- User chooses to create a new Configuration
- User specifies a name for the new configuration
- User specifies any/all of the following:
  - Default device
  - Default device/emulator parameters (for instance to configure maximum memory)
  - Associated source folders
  - Associated prerequisite projects
  - Conditional compilation symbol definitions
- User applies changes to accept new configuration

Select Project Configuration

- User opens project properties.
- User selects available project configuration for the project.
- User applies changes.
- Project is rebuilt based on new settings.

    - Preprocessing is done to match newly selected symbol definitions
    - Project libraries and source folders are set to match newly selected configuration
    - Standard JDT Compiler is invoked to build resulting source code

Edit Configuration

- User opens Configuration Management interface in workbench preferences
- User chooses a Configuration to edit
    - Double-click existing configuration or
    - Select configuration and press “Edit..” button
- User edits any/all of the following:
    - Default device
    - Default device/emulator parameters (for instance to configure maximum memory)
    - Associated source folders
    - Associated prerequisite projects
    - Conditional compilation symbol definitions
- User applies changes to accept configuration changes
- Projects associated with updated configuration will be rebuilt

Remove Configuration

- User opens Configuration Management
- User selects all configurations to remove
- User selects “Delete” key or “Remove” button
- User is prompted to confirm removal
- Configurations are removed
    - What happens to projects associated with this device?

Add Preprocessor Block to Source Manually

- User opens Java source code
- User enters appropriate preprocessor begin and end statements
- If referenced symbol is inactive/undefined, code between begin and end are automatically commented out

Add Preprocessor Block to Source via Menu

- User opens Java source code
- User selects block of code

- User opens context menu and selects Source -> Surround with #ifdef (or similar)
  - There could be any number of source editor context menu options to introduce conditional statements into the source
- Appropriate begin and end statements are added and block content is commented as appropriate

Export Ant Build Files

- User selects mobile project in Resource Navigator or Package Explorer
- User opens context menu
- User selects Mobile Tools -> Export Ant Build Files
- Build files are exported
  - Antenna support is one reasonable option here
  - Another option would be to use the headless antRunner support within Eclipse and a set of new tasks to tie into the Eclipse and Mobile Tools support

Execute Batch Build

- User selects mobile project in Resource Navigator or Package Explorer
- User opens context menu
- User selects Mobile Tools -> Batch Build...
- Batch build dialog is displayed showing all available configurations
  - When first opened, no configurations will be selected
  - When later opened, previously selected configurations will be selected
- User selects configurations to be built
- User selects “OK” button to launch batch build
- For each configuration selected:
  - Package will be built using configuration settings
  - Output will be placed in <project root>/build/<configuration name>