Build Message

We have a build process that does a lot of nested things. The build system should report errors in the process to the user: compilation problems, incorrect packages, etc.

We could have used exceptions but not only it would be hard to pass exceptions around, it also wouldn't be correct because users making mistakes is part of the expected use.

Also BuildMessage knows how to report errors better, recover on errors to collect as many errors in one pass as possible. For example, if several files fail to be parsed, we want to report all of them.

try/catch usually marks a code call and says "everything called from this try block, dynamically, is handled by this catch block". Build Message also has such property.

`buildmessage.capture` - takes a function and runs it in a buildmessage "context" and returns a message set (a set of errors). When the code is running inside that "capture", the errors from it go to this message set.

You can check if there were any errors by checking `messages.hasMessages()`.

There are different applications of `capture`, a lot of parts of the code wrap it. There is `captureAndExit` that exits immediately after printing. In catalog we have a wrapper that tries to run a function, if it has messages, refreshes the catalog and tries again.

In the lower-level code you usually never print the messages, you pass them up along the stack to be handled by something else (printing by cli or displaying by gui, etc).

`capture` is not useful without jobs. You use `buildmessage.enterJob` to create a new job. It has a title.

`buildmessage.error` is the function you use to say "something bad happened". The innermost job will provide more information to the error. Ex.: "While building package X: an error happened"

It should always be called inside a job. It will throw an exception if it is not inside, but you should not do this.

There is also a `buildmessage.exception`, the idea is, you use it to report the actual exceptions. Usually you use it to rethrow exceptions from a user package or a build plugin. It has some fancy stack-trace parsing and formatting. Ex.: "An exception while running your package.js file: TypeError ... ".

Another function that you would run a lot is `job.hasMessages()`. Unlike exceptions, when you notify the buildmessage of errors, it doesn't stop the execution, it keeps going. This gives you an opportunity to recover and go on to get more errors. But if you don't want to keep going, you can check if there

are any errors in the current job by calling `job.hasMessages()`. This call also includes messages from nested jobs.

Initially buildmessage was used only for error reporting, but now it is also used for the progress bars. It doesn't hurt the error reporting, it gives the error messages better locality (assuming it is useful).

Quick recap:

- `buildmessage.capture` - the same as a `try-catch` block
- `buildmessage.error` - is like throwing an exception (except it doesn't stop there)
- `buildmessage.enterJob` - is a concept similar to a stack-frame
- `capture.messages()` - get all errors
- `job.hasMessages()` - check if there are any errors so far

Misc

Helpers for asserting the code structure:

- `assertInJob`
- `assertInCapture`

A helper to run the user code:

- `files.runJavascript` - a helper to run the user code
- `buildmessage.markBoundary` - a helper that marks the entry point of the user code (like a build plugin), later when an `buildmessage.exception` is called, the buildmessage will remove unnecessary stack-frames from the stack-trace so the error displayed to the user doesn't contain any frames from tool.

Ex.:

```
var result = null;
f = function () { user provided code };
try {
  var markedF = buildmessage.markBoundary(f);
  result = markedF();
} catch (e) {
  buildmessage.exception(e);
  // clean up
  ...
  // pretend nothing happened so we can recover and move on to find other errors
  result = { name: "dummy" };
}
```