Handling Errors (v. 1.0)

C. S. Tritt, Ph.D.
December 6, 2011

Structured Programming

- About 35 years ago, programmers discovered that code was much more reliable and maintainable, if they restricted themselves to simple one-way in, one-way out loops and function calls/returns.
- This works fine until there is a serious error deep inside a complex program.
- In these exceptional cases, it is often best to provide another way out so the error can be correctly dealt with higher up in the program.

2

A Panic Button

- All modern programming languages provide similar "panic button" approaches to dealing with errors.
- This is generally referred to "throwing" an exception (or error).
- A thrown exception can be caught higher up in the program (outside of the loop or selection construct or in a calling function) or propagated further "up" the "stack".
- If an exception is not eventually caught, the program aborts and an error message is displayed.

3

What Functions Don't Know

- Most programs are interactive, but some still run in "batch mode."
- Functions typically don't know the context within which they are being called, so they can't know the best way to respond to an error.
- The calling program or function is in a much better position to know the appropriate response.
- Exceptions provide a way for functions to communicate errors to callers.

4

To Throw or Not to Throw

- Generally throw exceptions when the local code doesn't know how to deal with a situation.
- Generally throw exceptions under circumstances that are serious, rare and difficult (or impossible) to prevent.
- Acceptable circumstances include disks becoming full, programming errors (like not enough arguments), when a bad file name passed is into a function, etc.

5

More Information

- To learn more about the details of throwing and catching exceptions, see *Matlab > User's Guide > Programming Fundamentals > Error Handling*.
- Be careful. When you catch exceptions, you are interfering with the Matlab's normal error reporting mechanism and this can cause confusion.

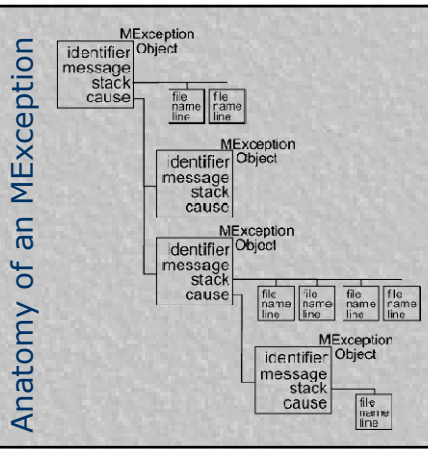
6

What Gets Thrown

- Matlab provides a special type of object, *MException,* for throwing exceptions.
- *MExceptions* have 4 fields:
  - *identifier*
  - *message*
  - *stack*
  - *cause*
- The purpose of the *MException* object is to store and transmit information about the error.

7

Anatomy of an MException



8

Identifier

- The identifier may not contain white space and always has the format: *component:mnemonic* where:
  - *component* indentifies broadest category of the source of the error (like *Matlab* or *Simulink*). It is often a good idea to start all your exceptions with a unique identifier (like *AcmeSoft*).
  - *mnemonic* is provides concise information about the error (like *TooFewArguments*).
- Exceptions are typically differentiated based on their *identifier*s.

9

Message

- The message is a more complete description of the exception and its cause. It should be written for whoever is expected to see it (the original programmer, some other programmer or the user).
- It can contain white space.
- An example would be "Field 'Accounts.clientName' not defined."

10

Stack

- The stack field of the *MException* object identifies the line number, function, and filename where the error was detected.
- It is populated automatically.
- Information on the entire chain of function calls leading to the point where the error occurred is stored in the stack.
- An example is shown on the next slide.

11

Cause

- In large, complex programs, it is sometimes useful to include additional, higher level information to exceptions as they "bubble up" through the calling functions.
- Use *addCause* function to add one or more lower level *MExceptions* to a higher level MException before it is thrown.
- Note that *cause* is a cell array.

12

Sample Code & MException

```
% ForcedError.m
%
% This script forces the generation of an MException.
%
% Created by Dr. C. S. Tritt
% Last revised: 12/5/11.

% Start with a clean workspace and command window.
clear all; clc;

try
   % Force an error.
   surf
catch myException
   % Do nothing. Just leave myException in the command window.
end
```

The try-catch block will be explained in a few slides.

13

Resulting Exception

```
>> myException

myException =

  MException

  Properties:
    identifier: 'MATLAB:nargchk:notEnoughInputs'
       message: 'Not enough input arguments.'
         cause: {0x1 cell}
         stack: [2x1 struct]

  Methods
```

14

Resulting Stack

```
>> myException.stack(1)

ans =

    file: [1x60 char]
    name: 'surf'
    line: 50

>> myException.stack(2)

ans =

    file:
'D:\classes\ge4200\TestCode\ForcedError.m'
    name: 'ForcedError'
    line: 14
```

15

The *error* function

- The simplest and oldest way to throw an exception is with the *error* function. It's syntax is:

```
error('msgIdent', 'msgString', v1, v2, ..., vN)
```

- The *error* function constructs an *MException* with the provided messages, automatically populates its stack and throws it.

16

try-catch Blocks

- Errors do not have to be caught.
- Errors that aren't caught cause the program to terminate and display an error message and/or code.
- *try-catch* blocks are use to define the response (remedial efforts) of the program to particular errors occurring in particular parts of programs as opposed to termination.
- Examples will follow.

17

MException Constructor

- Used to create an *MException*.
- More flexible than *error*.
- After creation, one or more causes can be added using the *addCause* function.
- The completed exception can then be thrown with *throw* (which fills in the stack data).
- See the *ForcedError2.m* handout.

18

Other Related Functions

- *getReport* – Display exception information in a nicely formatted way.
- *assert* – Used to test a specified condition during program execution and throw an exception if it is false.
- *rethrow* – typically used when the code in the catch block doesn't know how to handle the particular error caught.

19

An Example

- It is difficult to provide a completely realistic example of the use of *MException*, *try-catch*, etc. because these features are typically only really needed in very large, complex programs where the structured approach to programming makes ordinary error management too cumbersome.
- But I'll try anyway.

20

Example Summary

- See *ExceptionExample.m*
- This example demonstrates:
  - Dealing with exceptions across multiple levels of function calls.
  - Proper documentation of the exceptions a function can throw.
  - That catch code should identify and process only errors it knows about.
  - The use of *error, assert* and *rethrow.*
  - The use of *MException, addCause* and *throw*.
  - Argument number and type checking.
  - That both "built-in" and programmer generated exceptions can occur.

21