Configuration and Use of Stem

---

1. Stem

- **What is Stem?**
- **Architecture**
- **Configuration**
- **Running Stem**

- **Subsystems**

---

2. Why use Stem?

- **Convert Common Networking Code to Configurations**
- **Simplifies Complex Networking Code**
- **Easy to add Functionality through new Cell Modules**
- **No more dealing with sockets**
- **No more coding event loops**
- **Code Networked Applications at a Higher Level**
- **True Peer to Peer Networking**

---

3. What is Stem?

- **Network Toolkit and Application Suite**
- **Message Passing**
- **Event Driven**
- **Object Oriented**
- **Messages Can be Sent To/From any Stem Cell**
- **True Peer to Peer Communication**
- **Highly Modular and Scalable**
- **Pure Perl**

---

4. Stem Architecture

- **A Stem network is one or more connected Hubs (processes)**

- **Hub - An Event Driven Stem Process**
- **Hubs create and own Cells - An Addressable Stem Object**
- **Cells Communicate by sending Messages**
- **Dynamic control is supported by the Environment**
- **INSERT PICTURE HERE**

---

5. Stem Hubs

- **A Hub is single running Stem Process**
- **Hubs have unique names in a given Stem Network**
- **Can run multiple hubs on each CPU**
- **Hubs communicate via the Portal Class Cell**
- **Stem::Hub Class manages the Hub**

---

6. Stem Messages

- **Messages have header fields similar to E-mail**

  Each one ('to', 'from', or 'reply') has its own Address
- **Message Addresses are name triplets - Hub/Cell/Target**
- **Message Types and Fields**
- **Messages are delivered to Cells via special methods**
- **Stem::Portal Cells transfer Messages Between Hubs**

---

7. Stem Addresses

- **Name Triplet: Hub/Cell/Target**
- **Hub is Optional**

A message with no Hub address is first sent to current Hub or then to a default Hub.

- **Cell name is Required**

  It identifies which Cell in a Hub gets this Message

- **Target is Optional**

  It identifies which Cloned Cell in a Hub gets this Message or if that isn't found, the Cell with no target address is sent the Message. This is how a Parent Cell is addressed. Another Target use is a Cell which uses it internally as an identifier (see the Stem::Switch module).

---

8. Message Types and Fields

- **'to', 'from', 'reply' Attributes**

  Each can have a single Stem Address

- **'type' and 'cmd' Attributes**

  Sets the Message type and affects its Delivery

- **'data' Attribute**

  Holds a single reference to the data which can be any Perl Structure or object.

---

9. Message Delivery

- **Message Deliver is via Cell Methods**

  A Message of type 'foo' normally is delivered to a Cell method with the name foo_in. Common message types include 'data', 'stderr', 'response'. The delivery method in this Cell is called with the Message as its sole argument.

```
$method = "${type}_in" ;
$cell->$method( $msg ) ;
```

- **Command Message Delivery**

The special type 'cmd' means the Message has a 'cmd' attribute with a value (e.g. foo). This Message is delivered to the Cell via a method with the name foo_cmd. Also the return value of the method (if defined) is automatically sent back to the 'from' address in a 'response' type Message. This is used by many Stem commands.

```
my $method = "${cmd}_cmd" ;
my $response = $cell->$method( $msg ) ;
if ( defined $response ) {
        my $reply_msg = $self->reply( 'type' => 'response',
                                      'data' => \$response ) ;
        $reply_msg->dispatch() ;
}
```

- **Default Message Delivery**

  If no more appropriate method is found, the method 'msg_in' is used. If it doesn't exist, the Message is discarded and a Log entry is made with the unknown Cell Address

---

10. Stem::Portal Class

- **Stem::Portal supports sending Messages over sockets between Hubs**
- **Remote Cells are accessed as easily as local Cells**
- **Supports reconnecting after the pipe breaks.**
- **Has Secure Connections over ssh**
- **More Security Modules in Development**

---

11. Configuring Stem

- **How Configurations are Loaded**
- **Configuration Format**
- **Configuration Examples**
- **Supported by the Stem::Conf Module**
- **Table driven Attribute Parsing with Stem::Class Module**

---

12. Loading Configurations

- **File command line arguments to run_stem**
- **File names in a 'load' command Message - loads a remote config file**

- **'remote' command message - send a parsed config file to a Hub for execution**
- **stem_msg program can be used to inject a load configuration Message**

---

13. Configuration Format

- **All data is in Lists and Key/Value pairs**
- **[ ] demarks a list**
- **Keys are attribute and option names - single words**
- **Values can be any Perl values including other lists**
- **A config file has a set of Lists, each of which is one Class Configuration**

```
[
        class   =>      'Stem::SockMsg',
        name    =>      'D',
        args    =>      [
                port            => 6669,
                server          => 1,
                cell_info       => [
                        'data_addr'     => 'sw:d'
                ],
        ],
],
```

---

14. Configuration Examples

- **Simple Chat Server**

  chat.stem

- **Inetd Emulation**

  inetd.stem

- **Log File Monitor**

  monitor.stem

- **Log File Archiver**

  archive.stem

15. Stem::Cell Class

- **Supplies common services to Cells**
- **Supports Cloning of Parent Cells**
- **Supports data pipes to other Cells**
- **Used by Stem::SockMsg, Stem::Proc and other modules**

16. Stem Cells

- **Cell are registered Objects or Classes**
- **Cells send Messages to each other.**
- **Three Types of Cells**
- **Class Cell**
- **Configured Cell**
- **Cloned Cell**

17. Class Cells

- **System-wide resource**
- **Only Cell for any Class in a Hub**
- **Most Stem core Classes are Cells**

  Stem::Route, Stem::Portal, Stem::Log, etc.
- **Registered when the Class module is loaded (by the Stem core**

  or a Configuration)
- **A Class Cell can have Aliases**
- **Never has a Target address**

- **Example of Class Cell Registration of Stem::Vars**

```
Stem::Route::register_class( __PACKAGE__, 'var' ) ;
```

---

18. Configured Cells

- **Application Global Objects**
- **Many Cells can be created by a Class**
- **Created and Registered by a Configuration (local or remote)**
- **Can Be a Parent Cell and create Clones**
- **Never has a Target Address**
- **Examples:**

---

19. Clone Cells

- **Application Instance Objects**
- **Many Clones can be made from a Parent Cell**
- **Shares Cell Name with Parent Cell**
- **Managed by Parent Cell**
- **Always has a unique Cell/Target Address**
- **Uses Stem::Cell for clone support**
- **Examples**
- **Attribute description used by Class Constructor**

```
{
        'name'          => 'cell_info',
        'class'         => 'Stem::Cell',
},
```

- **Confguration enables Cloning**

```
[
```

```
        class   =>      'Stem::SockMsg',
        name    =>      'B',
        args    =>      [
                port            => 6667,
                server          => 1,
                cell_info       => [ 'cloneable' => 1 ],
        ],
],
```

---

20. Running Stem

- **Program is 'run_stem'**
- **Takes Stem Environment Arguments of the form: foo=bar**
- **Requires a list of configuration files to load and execute**
- **Starts up Stem - runs main event loop**

---

21. Stem Subsystems

- **Stem::Cron - Replacement for Unix cron**
- **Stem::Log - Replacement for syslog**
- **Stem::Debug - Creates Custom Trace/Debug subs for any module**
- **Stem::Cell - Cell Cloning and Pipe Support**

---

22. Stem::Cron

- **Supports cron-like Time Selections**
- **Supports Enhanced Time Selections - last day month, first weekday, etc.**
- **Only sends a Stem Message when triggered**
- **Message can be sent to one or more Cells (via a Switch module)**
- **One Stem::Cron can control entire Stem Network**
- **Can emulate Unix cron by triggering a Stem::Proc Cell**

23. Log File Management

- **Log Entry is Submitted to a Logical Log**
- **Logical Log has Filter Rules and Actions**
- **Rules test strings, numbers, Stem Environment**
- **Boolean Combinations of Rules and Actions**
- **Monitoring of External Logs**
- **Debug and Trace Subsystems Use Stem::Log**
- **Monitoring Subsystems Uses Stem::Log**
- **Syslog and Syslogd Replacement**
- **Local or Remote Log Files**

24. Trace/Debug

- **Modules can import multiple Trace/Debug subs**
- **Each sub can have custom name and default parameters**
- **Trace/Debug can be enabled by Stem Environment values**
- **Output goes into Stem::Log subsystem**

25. Stem Environment

- **Uses Global hash %Stem::Vars::Env**
- **Hash can be Imported into a Class as %Env**
- **Set from %ENV - STEM_BAR=foo becomes $Env{bar} = 'foo'**
- **Set with run_stem Args - bar=foo becomes $Env{bar} = 'foo'**
- **Set from Commands from STDIN when using Stem::TtyMsg**

- **Set with setenv command message sent to Class Cell Stem::Vars**
- **Used in Log Filters and by many other Stem Classes**
- **Used to set Class Attributes - 'env' option**

```
{
        'name'       => 'host',
        'default'    => 'localhost',
        'env'        => 'host',
}
```