

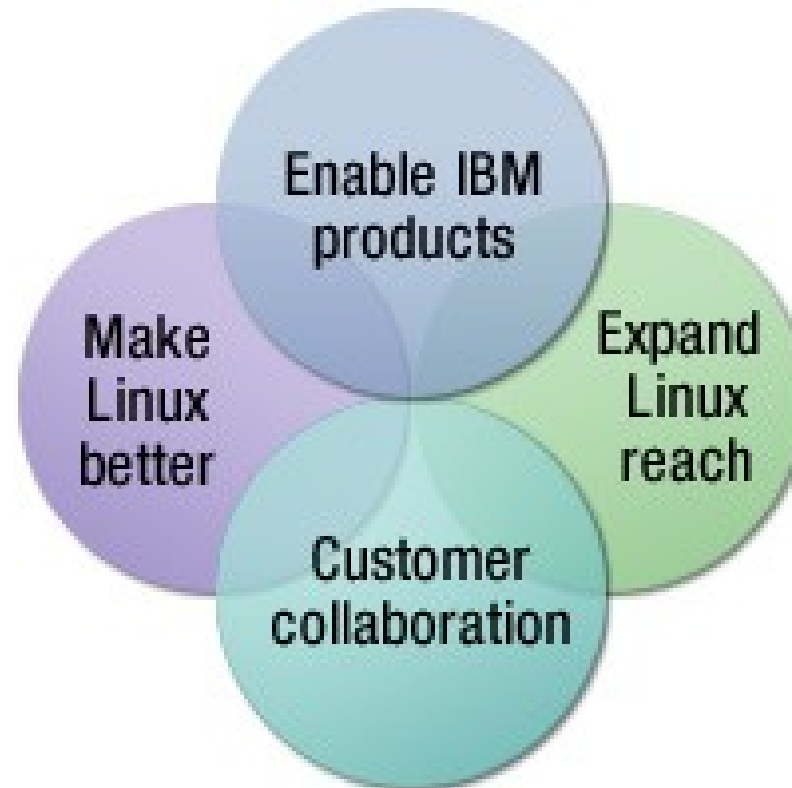


Linux Technology Center

# Remote Tools Plug-in

Daniel Felix Ferber  
[dfferber@br.ibm.com](mailto:dfferber@br.ibm.com)

# About Linux Technology Center (LTC)



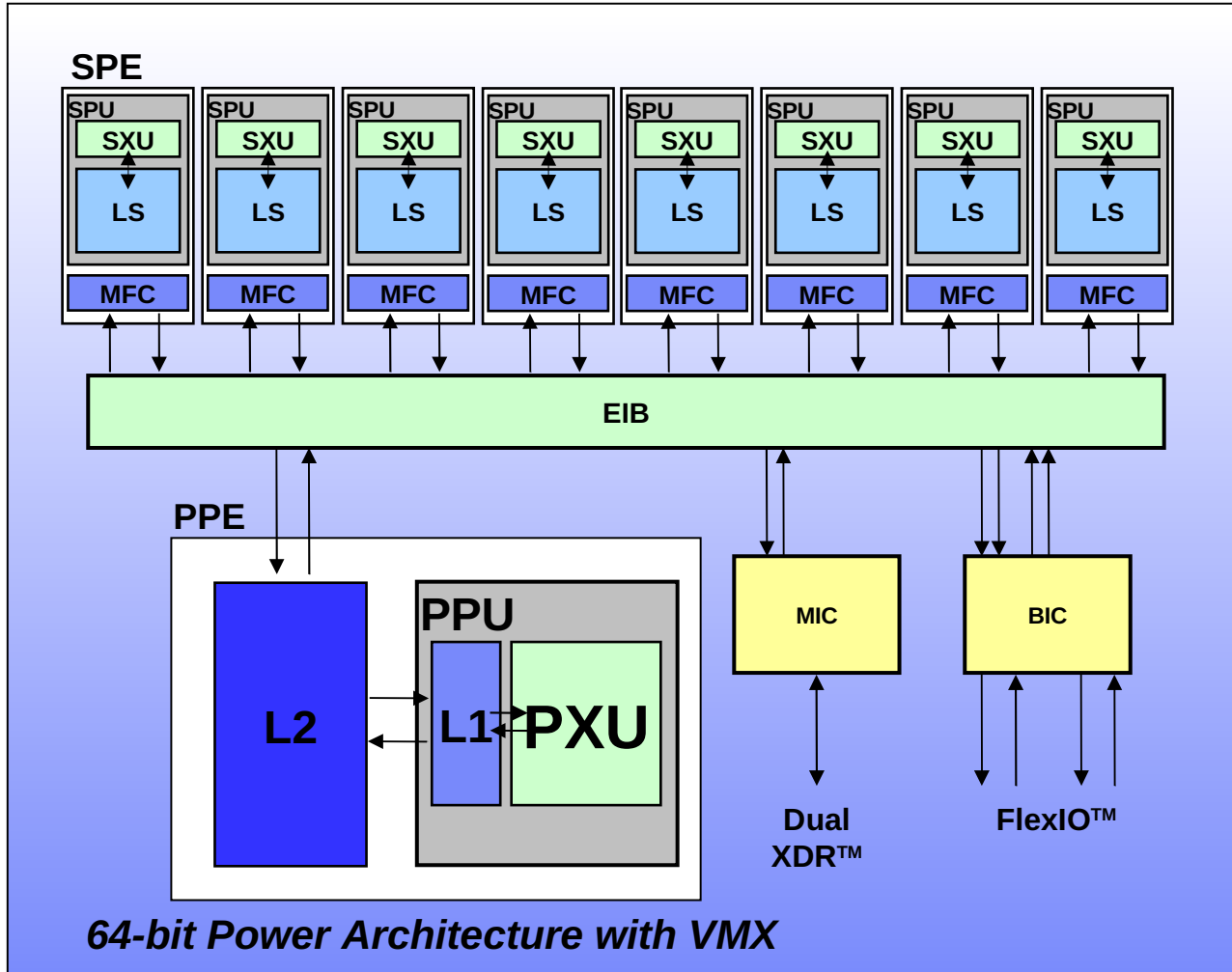
**Collaborate → Innovate**

The Linux Technology Center (LTC) is an IBM team of open source software developers who work in cooperation with the Linux open source development community. The LTC serves as a center of technical competency for Linux.

# About IBM Integrated Development Environment for Cell Broadband Engine SDK

- A set of Eclipse plug-ins that integrate the Cell Broadband Engine tool chain and enable rapid building of Cell Broadband Engine applications.
- Features:
  - a C/C++ editor that supports syntax highlighting; a customizable template; and an outline window view for procedures, variables, declarations, and functions that appear in source code
  - a rich visual interface for PPE (Power Processing Element) and SPE (Synergistic Processing Element) GDB (GNU debugger)
  - seamless integration of simulator into Eclipse
  - automatic builder, performance tools, and several other enhancements.

# About Cell BE & Cell SDK



## Cell SDK 3.0

- <http://www.alphaworks.ibm.com/tech/cellide>
- <http://www.ibm.com/developerworks/power/cell/>
- <http://www.alphaworks.ibm.com/tech/cellsw>
- <http://www.ibm.com/developerworks/power>

## Objectives

- To provide a Java library and Eclipse plug-ins for secure, platform independent access to remote Linux hosts.
- Current focus on SSH (Secure Shell) protocol.

## Other motivations:

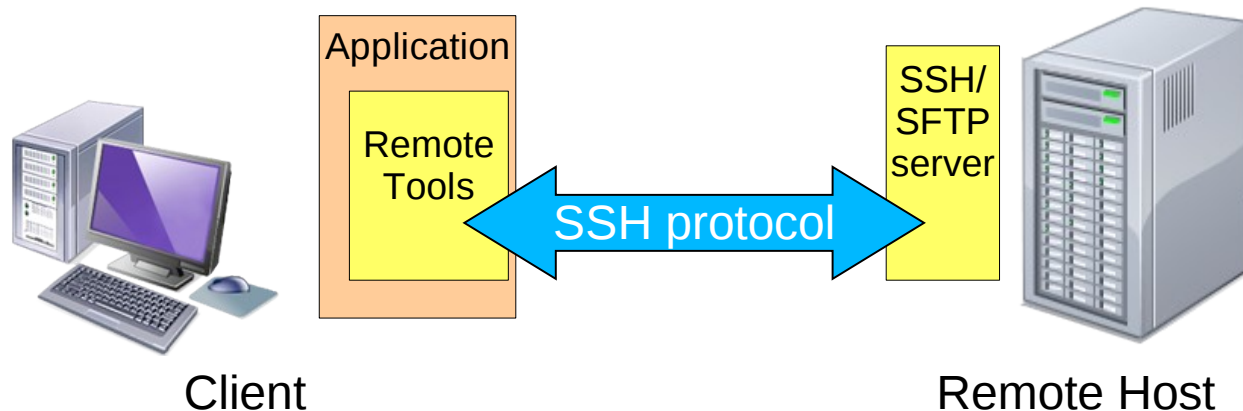
- Provide plug-ins for Cell IDE for features like:
  - Cell BE Simulator integration.
  - Launch Cell BE applications on simulator target.
  - Launch Cell BE applications on Cell blade target.
  - Remote debugging.
- All targets allow SSH connections.

## Solution

- A Java library that connects to a remote host using SSH protocol and hides it's complexity.
  
- High level plug-ins for Eclipse:
  - Remote application launcher framework.
  - Remote target environment manager.

## How?

- Remote Tools connects and authenticates to remote host.
- Local method calls are executed as Bash scripts on remote host.
- SFTP for file transfer and remote file system navigation.



## Usage scenario

- Remote access through firewall/NAT
  - Use Remote Tools library to connect, authenticate on remote host. Then forward ports from/to the internal network.
- Execution of tools on remote host
  - Use Remote Tools library to copy files, configure environment and start process on the remote host.
- Profiling applications
  - Extend Remote Launcher framework with profiling features to measure the application performance on the remote host.

## Why SSH?

- Available as default on all Linux distros.
  - OpenSSH: open source, free software, LGPL
- Does not require installing extra software on host or client.
- Secure authentication and connection.
- Reasonable performance.
- Implementation of most important features:
  - Execution, port forwarding, file system access (sftp), X11 forwarding
  - Allows running bash scripts.
- Firewall friendly.

## Remote Tools Library

## Why Remote Tools library

- Many low level SSH client libraries:
  - Ganymed SSH-2 for Java, J2SSH Maverick, JSSH, **JSch**
- Not open source, or not contributed to Eclipse
  - IBM Tivoli Remote EXecution and Access (RXA), IBM Secure Shell Library for Java.
- Specific set of required features
  - Port forwarding, PTY, X11 forwarding.
- Depend only on Linux standard configuration.

## Remote Tools Library

- Features:
  - Execution of Bash scripts
  - Execution of Linux process
  - File system navigation
  - File transfer (upload/download)
  - Port forwarding (remote/local)
  - Remote status queries
  
- Based on JSch – Java Secure Channel
  - JSch is a implementation of SSH and SFTP client.
  - <http://www.jcraft.com/jsch/>

## Why Jsch?

- Native Java implementation.
- Free software, open source, EPL.
- Distributed within Eclipse Platform.
- Active support and development.
- Stable, conforms to standards.

## Why encapsulate JSch?

- JSch disadvantages:
  - Only provides low level operations.
  - Exposes SSH protocol details.
  - Requires expert SSH knowledge to write code.
  - No documentation: only SSH specification.
  
  - Class hierarchy is confusing.
  - Considerable coding effort with JSch.
  - Difficulty to handle errors.

## Why encapsulate JSch?

- Remote Tools advantages:
  - Reduce programming efforts.
  - Easier error handling.
  - Better organization of features sets.
  
  - Abstract interfaces, easy to use.
  - High level operations: queries, file transfer.
  - JavaDoc.

## Example with JSch

```
JSch jsch=new JSch();
Session session=jsch.getSession(user, host, 22);
session.setX11Host(xhost);
session.setX11Port(xport+6000);
UserInfo ui=new MyUserInfo();
session.setUserInfo(ui);
session.connect();
```

```
Channel channel=session.openChannel("exec");
((ChannelExec)channel).setCommand(command);
channel.setInputStream(null);
((ChannelExec)channel).setErrStream(System.err);
InputStream in=channel.getInputStream();
channel.connect();
```

```
byte[] tmp=new byte[1024];
while(true){
    while(in.available(>0){
        int i=in.read(tmp, 0, 1024);
        if(i<0)break;
        System.out.print(new String(tmp, 0, i));
    }
    if(channel.isClosed()){
        System.out.println("exit-status: "+channel.getExitStatus());
        break;
    }
    try{Thread.sleep(1000);}catch(Exception ee){}
}
```

```
channel.disconnect();
session.disconnect();
```

## Example with Remote Tools library

```
RemotetoolsPlugin remoteTools = RemotetoolsPlugin.getDefault();  
PasswdAuthToken token = new PasswdAuthToken(user, password);  
IRemoteConnection connection = remoteTools.createSSHConnection(token, host);  
connection.connect();
```

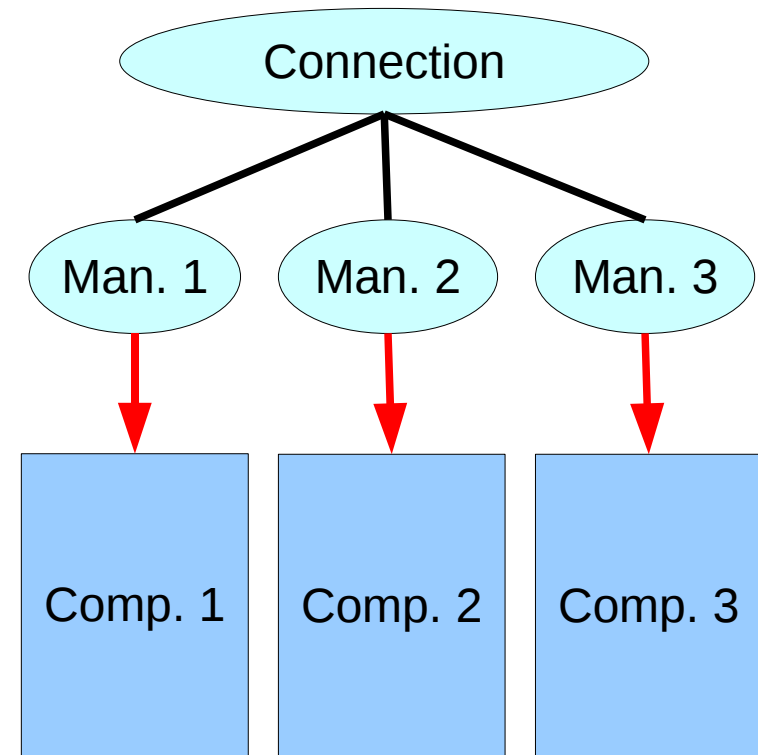
```
IRemoteExecutionManager manager = connection.createRemoteExecutionManager();  
IRemoteExecutionTools tool = manager.getExecutionTools();  
IRemoteScript script = tool.createScript();  
script.setProcessErrorStream(System.err);  
ByteArrayOutputStream stream = new ByteArrayOutputStream();  
script.setProcessOutputStream(stream);  
script.setScript(command);
```

```
Process process = tool.executeProcess(script);  
process.waitFor();  
System.out.print(stream.toString());  
System.out.println("exit-status: "+process.exitValue());
```

```
manager.close();  
connection.disconnect();
```

## Remote Tools library model

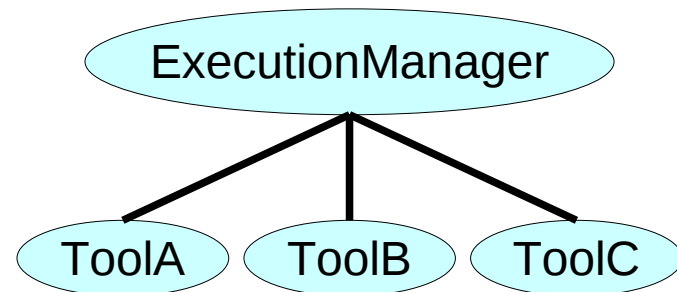
- Execution manager
  - Allows simultaneous operations
  - Managers isolate components
  - Encapsulation
  - Reliability
  - Reduced coupling
  - Scalability



Allows multiple independent components to contribute to a common task.  
Eg.: Remote Application Launcher Framework!

## Remote Tools library model

- Authentication Token:
  - User name/password.
  - Private/public key.
- Copy Tools:
  - (recursive) upload/download of files.
- Execution Tools:
  - Scripts/processes.
- Status Tools
  - User, group, time, used sockets.
- File Tools
  - File system view.
- Forwarding Tools
  - Local/remote port forwarding.
- Path Tools
  - Split, join paths using SSH protocol path syntax.



## Remote Tools library internals

- Simultaneous channels.
- No Bash initialization.
- Lightweight cypher
- Connection Pool/PTY limitation.
- Path conversion/quoting.
- Bash script performance & /proc readings.

## Exceptions in Remote Tools library

- RemoteConnectionException
  - Connection, authentication, unexpected drop.
- RemoteExecutionException:
  - Processes, scripts, file uploads/downloads.
- RemoteOperationException:
  - File system navigation, file attribute changes.
- PortForwardingException:
  - Port forwarding, X11 forwarding

Unexpected

Expected/Predictable

Clear separation of expected/possible from unexpected rare errors.  
Simplifies programming.

# Remote Application Launcher Framework

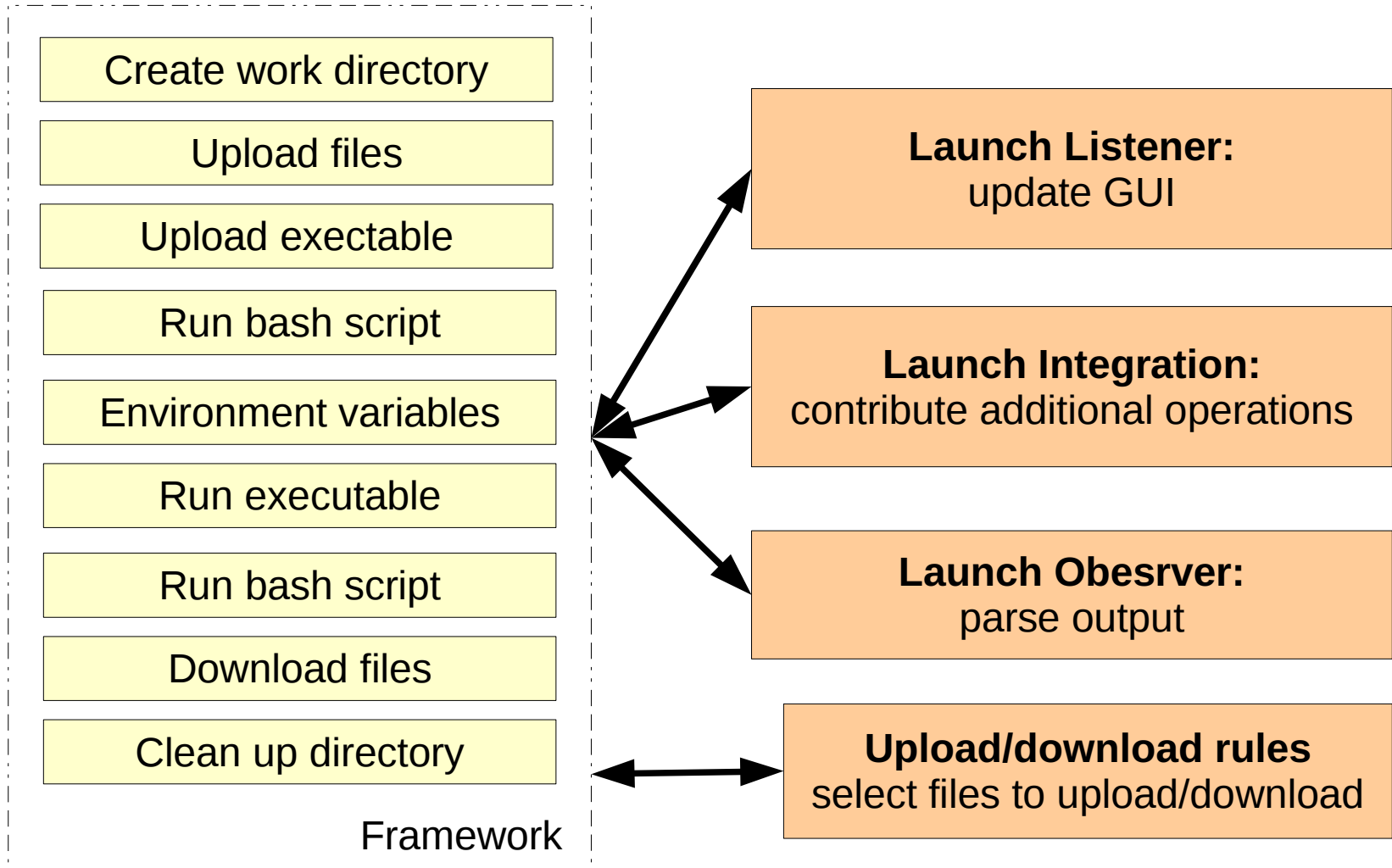
## Remote Application Launcher Framework

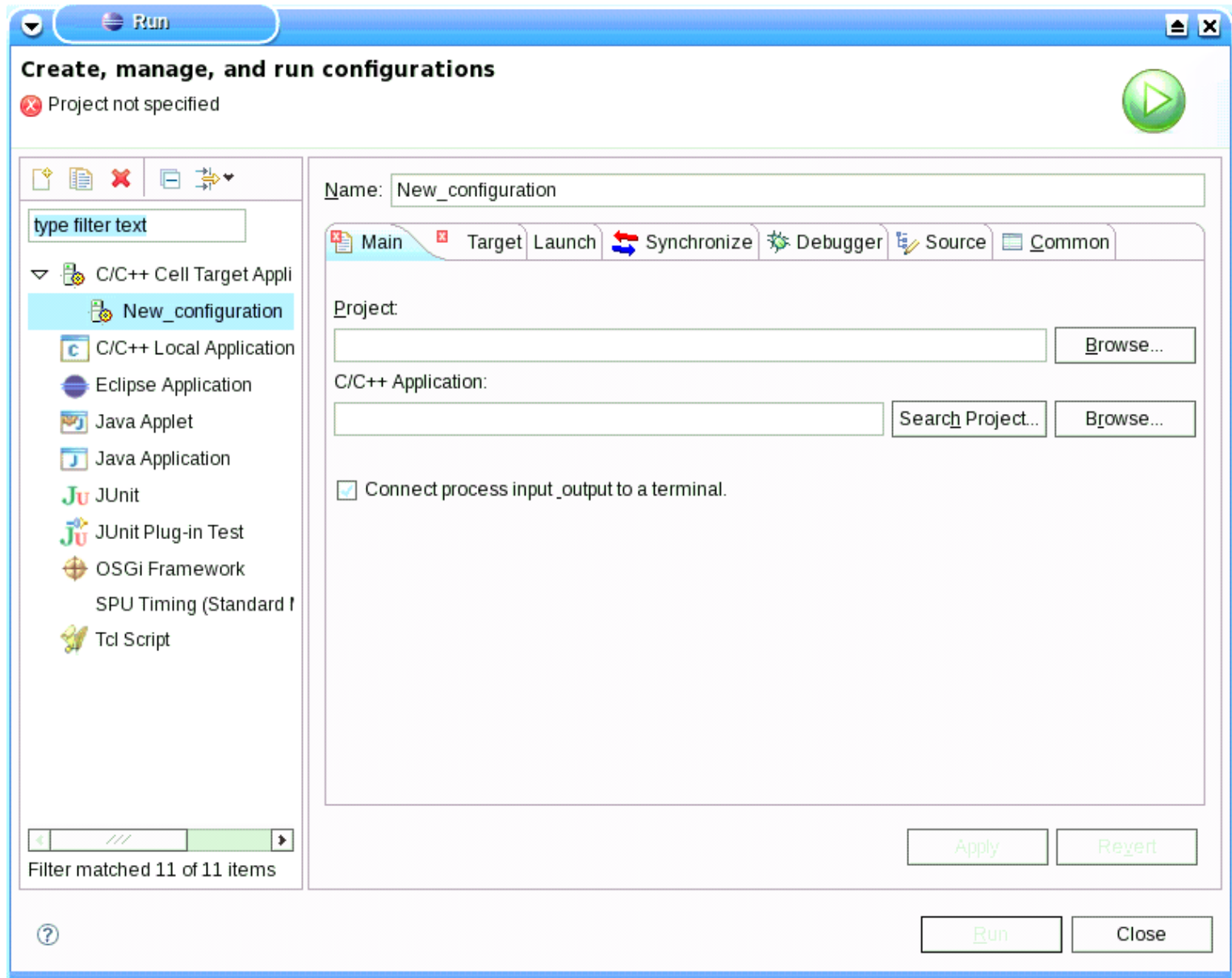
- Launches an executable on a remote host.
- Implements a skeleton to launch the application with Remote Tools library.
- Tools extend framework to provide customized functionalities.
- Extends CDT C/C++ application launcher.

## Remote Launcher features

- Stdin/stdout/stderr in Eclipse console.
- Extension point to parse stdout.
- Delegate contribution of customized operations.
- Bash script to configure remote host.
- Temporary working directory.
- Upload executable, libraries, dependencies.
- Upload input data, download output data.
- Remote environment variables.
- Allows Eclipse debugger integration.

# Remote Launcher framework





# Remote Target Environment Manager

## Remote Target Environment Manager

- Manages a set of remote hosts
  - Real hosts: Linux machines
  - Virtual hosts: Simulator, emulator
- Allows launching applications on these targets
- Stores configurations that describe a remote host
- GUI to allow user to create/edit targets.
- View to control each target (connect/disconnect)

Problems Tasks Console

Target Environment

- Attached Cell Simulator
- Generic Host**
  - 192.168.10.11
  - Local Cell Simulator
- Remote Cell Simulator
  - Remote Cell Simulator

### Target Environment Configuration

#### Generic Host

Properties for connecting to a generic host

Target name: 192.168.10.11

Host Information

Localhost  Remote host

Host: 192.168.10.11 Port: 22

User: dfferber

Password based authentication

Password:

Public key based authentication

File with private key:

Passphrase:

Cell application launch:

Base directory: /tmp/remotetools

Status

Stopped

Stopped

## Conclusion

- Remote Tools is a simple, but comprehensive and powerful open source solution to enable Eclipse to launch applications and access files on a remote host.

## Where to get

- Plugins from Cell IDE
- Contributed to PTP - Parallel Tools Platform
  - <http://www.eclipse.org/ptp/>