Remote access with the DSDP Target Management Project

Martin Oberhuber, Wind River

www.eclipse.org/dsdp/tm
Tutorial Themes

How we’re going to run this:

• Practical
• Interactive
• Workspace Take-away
What do you need?

• See org.eclipsecon.tmtutorial.docs/01_prerequisites.html

• Required stuff (approx. 20MB)
  - This presentation
  - Example plugins and docs (org.eclipsecon.tmtutorial)
  - TCF code and examples
  - RSE-SDK and example projects

• Optional stuff (approx. 230MB)
  - Qemu with Linux image
  - DSF-SDK-N20071113-0200
  - TmL Demo

• Base Downloads (350MB) (Eclipse, CDT, Subversive)
Interactive: Who are you?

- Name
- Affiliation
- What you want to do with TM / RSE
- Tutorial Expectations

- Drop RSE-SDK*.zip (and optionally DSF-SDK*.zip) into your Eclipse
- Extract all example ZIPS (tmtutorial.zip, tcf.zip / optionally .metadata.zip, tmldemo.zip) into the same new .../workspace folder
- Either import all plugins into existing workspace, or open the new workspace
What we’ll do

• The TM Big Picture
• Target Communication Framework (TCF)
  ◦ Concepts and Architecture
• TCF Sample Session
  ◦ Command-line client
  ◦ Value-add
  ◦ RSE Integration
  ◦ Future, Resources, Pointers and Getting Involved
What we’ll do (Continued)

• Remote System Explorer (RSE) Part I: Client Use
  ♦ Ex.1: Working with the SystemRegistry - Creating a Host
  ♦ Ex.2: Working with Events – An Event logging view
  ♦ Ex.3: Working with Actions and FileServiceSubsystem - Upload
  ♦ Ex.4: Remote Command Execution – nm

• RSE Part II: Extending RSE (Subsystems and Filters)
  (Official RSE Examples, explained)
  ♦ Ex.6: Registering a Custom IFileService – FTP
  ♦ Ex.7: A Custom Subsystem with Filters – Developer

• Examples for Commercial Adoption of RSE
• What’s currently brewing – upcoming changes and news
• Resources, Pointers and Getting Involved
• Q&A
System Debug: the Big Picture

Eclipse

3rd party
CPU Debugger

3rd party
DSP Debugger

Target Description
IP-XACT, XML

Target Server(s)

SoC (Model)

CPU

HW

IP

DSP

TCF

TCF

TCF

TCF
Target Management 3.0 Components

- **Remote System Explorer (RSE)**
  - Wizards
  - Views
  - Reusable Widgets
  - Filters
  - Subsystems & ElementAdapters
  - Persistence
  - Services (Files, Processes, Shells)
  - Data Models (SystemType, SystemRegistry)

- **Terminal & Discovery integrations**
  - Terminal
    - view
    - connectors
    - widget
  - Discovery
    - view
    - protocols
    - model

- **TCF Core + Services**
- **dstore**
- **Standard Protocols (ssh, ftp, zeroconf…)**

**Diagram Notes**
- **Eclipse Platform**
- **TCF Integrations**
  - DSF
  - CDT
- **Platform only**
- **RCF only**
- **Widget**
- **EMF**
TCF - Background

• Cross development tools need communication
  ❖ Many tools, each typically using its own agent and communication method
  ❖ Lots of overlap between these, e.g. how to communicate, retrieve/model target objects, manipulate target, etc
Motivation

• Almost every cross development tool have their own infrastructure (agent, connection, protocol, setup, etc)

• This leads to:
  ❖ Poor user experience
    ▪ Each tools has its own target configuration
    ▪ Increased target intrusion (footprint, multiple agent interaction)
    ▪ Inconsistent product availability matrix
  ❖ No sharing between agents
    ▪ Duplicated maintenance effort
    ▪ New features have to be added in multiple places
    ▪ New tools have to start from ground zero
  ❖ Limited Eco-system
TCF - Outline

• Define an open end to end tool to target communication mechanism for development, debug, monitor, analysis and test

• Specification
  ▶ Transport channel supporting extensible set of “services”
    ▶ Typically on top of a TCP/IP stream, but other transports supported as needed but the target
  ▶ Services defining commands, progress, replies, events & semantics
  ▶ Discovery of available servers and services

• Prototype implementation
  ▶ Eclipse plug-ins
  ▶ C-based agent

• Scope
  ▶ Cross tools (i.e. host and target are different) benefits the most, but is applicable to native tools as well
  ▶ Target agent, OCD/JTAG and simulator connections
TCF - Core Design Ideas

• Service knows best how to represent the system – get information from there and data-drive layers above
  ◆ If not possible, put the knowledge in the lowest possible layer and data drive the layers above
• Use the same protocol end-to-end, but allow value-adding servers to intercept select services when needed and pass-through everything else
• Services as building blocks that can be used by multiple clients (tools) for different environments (target agent, OCD, simulator)
  ◆ Avoid tools specific agents
  ◆ Bridge gap with environment specific services to setup/configure common services
• Support high latency communication links
Architecture Overview

UI

Tool A

Tool B

Tool C

Tool D

Host

Service 4

Value Add

Service 5

P1

Target

Service Manager

Service 1

Service 2

Service 3

P1
Use Case: SimpleJtagDevice

• Debug (run-control, breakpoint, memory register)
• Possibly Others (flash programming, download, etc)
Use Case: TestExceutionAgent

• Process launch and kill
• Standard I/O redirection
• File system access
Use Case: LinuxUserModeAgent

• Debug (run-control, breakpoint, memory, register)
• OS Awareness (process/thread list, CPU utilization, etc potentially with value-add)
• Process launch and kill
• Standard I/O redirection
• File system access
• Monitoring (event-config, event-log)
Specification Status

• Transport Channel
• Current Services
  - Run Control, Memory, Register, Breakpoint, Processes, Stack Trace, File System, System Monitoring

• TCF is a Protocol independent of API. ECF is an API independent of Protocol.

• Review of current and specification of additional services in power.org and Eclipse
TCF Sample Session

• Extract `puppy_tcf.zip` (see 01_prerequisites.html for creation)
• Run `puppy_redir.bat` (Windows) or `puppy_redir.sh` (Linux)
  - Launches QEMU + Puppy Linux, with TCF pre-built
• Open Console 1 for agent:
  - `cd /root/org.eclipse.tm.tcf.agent && ./agent -L-`
• Open Console 2 for client:
  - `cd /root/org.eclipse.tm.tcf.agent && ./client -L-`
  - `peers`
  - `connect tcp:127.0.0.1:1534`
  - `tcf FileSystem roots`
  - `tcf FileSystem opendir "/root"`
  - `tcf FileSystem readdir "FS0"`
  - `tcf Processes getChildren "" false`

Logging to stdout
Agent is auto-detected
JSON messages Using an ID
Connecting QEMU from outside

• This is in puppy_redir.bat:
  - start puppy.exe -redir tcp:1534::1534

• 1534 is the TCF default port for discovery. QEMU forwards it from the client to the host in both directions

• From Eclipse, launch RSE+TCF
  - Run > Debug Configurations > Eclipse App
  - Open RSE Perspective
  - New Connection : TCF
  - Expand Processes / All Processes

→ Shows QEMU Linux Processes
Debugging

• Run > Debug Configurations > TCF
  ◦ Select connection (auto-discovered)
  ◦ Program: /root/helloworld/helloworld
  ◦ Args: “tcf is cool”
  ◦ Debug
• Switch to “Debug Perspective”
• Show View “TCF Trace”
• Suspend / Resume, Registers
TCF Service Implementation

- Asynchronous: `DoneMkDir` is the Callback
- Commands are put into a queue to be run on Command Thread
TCF Plugins

• Org.eclipse.tm.tcf – Core Java framework
• Org.eclipse.tm.tcf.agent – The agent (plain C)
• Org.eclipse.tm.tcf.debug.* - Debug Integration
• Org.eclipse.tm.tcf.docs –
• Org.eclipse.tm.tcf.dsf.* - DSF integration
• Org.eclipse.tm.tcf.examples.daytime.* - How to create a custom Service (both agent and client)
• Org.eclipse.tm.tcf.rse – RSE Files and Processes
A value-add example

• Run on QEMU:
  • Shell 1: ./agent –L-
  • Shell 2: ./valueadd –L- -sTCP:127.0.0.1:12345
  • Shell 3: ./client –L-
    • peers
    • connect TCP:127.0.0.1:12345
    • tcf FileSystem roots
    • connect TCP:127.0.0.1:12345
    • tcf Locator redirect “TCP:127.0.0.1:1534”
    • tcf FileSystem roots
TCF: Next Steps

• We need YOU getting involved!
  - mailto:dsdp-tm-dev@eclipse.org
  - Bugzilla, Newsgroup
  - Your requirements and ideas?
  - Many things to discuss with respect to Context Specification, Debug Model, Services

• Currently planned next steps
  - Port DSF integration to DSF HEAD
  - Basic Debugging Services on Windows agent
  - Yes Wind River is going to adopt this!
Links

• Prototype source repository
  - http://www.eclipse.org/dsdp/tm/development/tcf-anonymous.psf
  - svn://dev.eclipse.org/svnroot/dsdp/org.eclipse.tm.tcf/trunk
  - http://dev.eclipse.org/viewsvn/index.cgi/org.eclipse.tm.tcf/?root=DSDP_SVN

• FAQ
  - Has links to all Documentation:
    - Getting Started (less than what we did)
    - Protocol Specification (messages, events, JSON)
    - Services description
    - Agent description
Questions Regarding TCF?
And now for something completely different…

• Remote System Explorer (RSE)
  ✷ A consistent UI for anything remote
  ✷ Needs to handle long delays and connection errors
  ✷ Everything is done in a Job
  ✷ Concept of SystemTypes, Subsystems
RSE Model Objects, part I (Connections)

Model

«extension point»
IRSEType
(The static type; also seen in the New Connection Wizard)

IHost
(The actual Instance)

Stored in
IProfile
(not shown here)

«extension point»
ISubSystem
(The static type)

ISubSystem
(The actual Instance)

Controller

IProfile
Registering types, configurations and instances; dispatching events

All these elements are meant to be non-UI (ISubSystem* not yet: bug 170923)
Ex. 1: Programmatically creating a connection

• Goal: Have a toolbar button for creating an ssh connection to “build.eclipse.org” (which will be used by tooling later on)

• Tasks:
  • Use PDE Tools to create a plugin from “Hello World” sample (this does the button for you)
  • In the button’s run() method,
    ▪ Get the ISystemRegistry from the RSECorePlugin class
    ▪ From the Registry’s Profile Manager, get the default profile
    ▪ Ask Registry if host “build.eclipse.org” is already there
    ▪ If not, create it: System Type=“SSH Only”

• Solution:
  • org.eclipsecon.tmtutorial.host.CreateEclipseHostActionDelegate
Ex. 1: Creating a Connection - Solution

```java
public void run() {
    String hostName = "build.eclipse.org";  //NON-NLS-1$
    ISystemRegistry registry = RSECorePlugin.getDefault().getSystemRegistry();
    ISystemProfile profile = registry.getBytesystemProfileManager().getDefaultPrivateSystemProfile();
    IHost host = registry.getHost(profile, hostName);
    if (host == null) {
        host = registry.createHost("SSH Only", //System Type Name
                                   hostName,  //Connection name
                                   hostName,  //IP Address
                                   "Connection to Eclipse build site"); //description
    }
}
```
RSE Model Objects, part II (PropertySets)

Property Sets are non-UI. IConnectorService not yet (bug 170923, again). Most RSE Model Objects can have Property Sets.

Model

IConnectorService
- Created by ISubSystemConfiguration when registered against an IHost
- Resembles a physical connection to an IHost, e.g ssh
- Maintains state for it (connected / disconnected, asks for password)
- Can be shared by multiple ISubSystems (e.g. files, shells)
- When one IHost has multiple subsystems, there can be multiple connector services.

PropertySet
Generic container for data

PropertyType
Meta-Info (String, Integer, Boolean); not shown here

Property
A name/value pair
Ex. 1a: Storing Custom Properties

• Goal: Store the well-known architecture of “build.eclipse.org” with the connection (for informational purpose).

• Tasks:
  - In Example 1 button’s run() method, after creating the IHost,
    - Find the connection’s first IConnectorService
    - Create a new PropertySet (“System Info”)
    - Add a new Property “Arch” with contents “PPC64”

• Solution:
  - org.eclipsecon.tmtutorial.host.CreateEclipseHostActionDelegate
Ex. 1a: Storing Properties - Solution

// example of using property sets
IConnectorService[] conServices = host.getConnectorServices();
if (conServices != null && conServices.length > 0) {
    IPropertySet set = null;
    IPropertySet[] sets = conServices[0].getPropertySets();
    if (sets != null && sets.length > 0) {
        set = sets[0];
    } else {
        set = new PropertySet("System Info");
        conServices[0].addPropertySet(set);
    }
    set.addProperty("Arch", "PPC 64");
}
RSE Model Objects (part IIIa)

- **RSE Resources**: YOUR model objects or re-using existing subsystem, e.g. IRemoteFile

- **Model**: Presentation of YOUR elements, Options for Actions, menus, ...
  - Created by an AdapterFactory, which is registered with Eclipse by your Plugin Activator.

- **View**: «interface» IAdaptable

- **YourResourceElementAdapter**: «interface» ISystemViewElementAdapter
RSE Events are for Resources. Resources below a Subsystem are unknown “Objects” of some contributed model. Adapting them to ISystemViewElementAdapter gives the most important Properties, which are also shown in the view.
Ex. 2: An RSE Event Logging Console

• Goal: Register for all RSE Events, and display them as Text in a Console (for debugging purpose). Your applications could use events e.g. to do cleanup after a Filter is deleted.

• Tasks:
  ✷ Create an instance of RSE Event Listener, which prints to a Console
  ✷ Register the Listener with the ISystemRegistry (could be done on startup of Workbench)

• Solution:
  ✷ org.eclipsecon.tmtutorial.eventconsole.RSEEEventLogging
The Event Logging Console

- Window > Show View > General > Console
- Watch RSE Events being generated as you work
Ex. 2: Event Logging - Solution

```java
public void systemResourceChanged(ISystemResourceChangeEvent event) {
    int type = event.getType();
    String eventStr = getResourceChangeEventType(type); // int to String
    if (resource instanceof IAdaptable) {
        ISystemViewElementAdapter adapter = (ISystemViewElementAdapter)
            ((IAdaptable)resource).getAdapter(ISystemViewElementAdapter.class);
        if (adapter != null) {
            String type = adapter.getType(resource);
            String name = adapter.getName(resource);
            String message = eventStr + "\t(" + type + ")\t" + name;
            logEvent(message); // print into Console; could also do stdout
        }
    }
}
```
RSE Tools for Remote Files

Model
- IRemoteFile
- IRemoteFileSubSystem
- IHostFile
- IFileService

Subsystem
- SystemRemoteFileDialog
- SystemRemoteFolderDialog

Service
- WorkspaceResourceSet
- UniversalFileTransferUtility

Controller
Why are there Subsystem and Service layers?

• Originally, RSE just dealt with Subsystems
  ♦ You can register just ANYTHING as a Subsystem.

• It turned out, that some **Subsystems should be used with multiple protocols** (e.g. files-via-dstore, files-via-ssh, files-via-ftp)
  ♦ The Service Layer allows to replace the protocol
  ♦ UI code, filters, widgets etc. are re-used from the Subsystem

• The Subsystem is the client-facing side (filters, dialogs, …) although it has both a non-UI layer and a UI layer (via Adapters).

• The Service is always non-UI. It’s for programmers.

• For your own subsystem, you can but don’t have to do a Service.
Ex. 3: FileServiceSubSystem - Upload

• Goal: Register a context menu action that’s valid on any IResource in Eclipse Resource Navigator. When invoked, show a dialog prompting for a target location on “build.eclipse.org”, and upload.

• Tasks:
  ✷ Use PDE, New Plugin, popupMenu Wizard to create action
  ✷ In run() method:
    ▪ Get IHost for “build.eclipse.org” from system registry
    ▪ Use SystemRemoteFolderDialog to prompt for upload folder
    ▪ Create a SystemWorkspaceResourceSet as source
    ▪ Use UniversalFileTransferUtility.copyWorkspaceResourcesToRemote()

• Solution:
  ✷ org.eclipsecon.tmtutorial.jarsigning.JarSigningActionDelegate
Ex. 3: Upload - Solution

// reusable RSE dialog for browsing folders of remote systems
SystemRemoteFolderDialog dlg = new SystemRemoteFolderDialog(
    shell, "Select Location", theHost);

int result = dlg.open();
if (result == Window.OK) {

    Object output = dlg.getOutputObject(); // get the selected item
    if (output instanceof IRemoteFile) {

        IRemoteFile targetFolder = (IRemoteFile)output;

        SystemWorkspaceResourceSet workspaceSet = new SystemWorkspaceResourceSet();
        for (int i = 0; i < _selectedFiles.size(); i++) {
            workspaceSet.addResource(_selectedFiles.get(i));
        }

        SystemRemoteResourceSet results =
            UniversalFileTransferUtility.copyWorkspaceResourcesToRemote(
                workspaceSet, targetFolder, monitor, false);

        targetFolder.markStale(true); // refresh parent (if applicable in ui)
        registry.fireEvent(new SystemResourceChangeEvent(targetFolder,
            // fire refresh
            ISystemResourceChangeEvents.EVENT_REFRESH, targetFolder));
    }
}
Ex. 3a: Jar signing on build.eclipse.org

• Goal: After uploading a jar file, invoke the “sign” script on build.eclipse.org and wait for the result to appear. Then, download it again.

• Tasks:
  ♦ Using previous upload example, after uploading
  ♦ In run() method:
    ▪ Compute the target folder for signing
    ▪ Get IRemoteCommandSubSystem to run “sign”
    ▪ Poll the target folder until the output is there
    ▪ Download the output

• Solution:
  ♦ org.eclipsecon.tmtutorial.jarsigning.JarSigningActionDelegate
Ex. 3a: Jar signing - Solution

//Create folder for output
IRemoteFile parent = jarToSign.getParentRemoteFile();
IRemoteFile outdir = fileSS.getRemoteFileObject(parent, "rseout");
if (!outdir.exists()) fileSS.createFolder(outdir);

//ensure the target does not exist yet
IRemoteFile outputFile = fileSS.getRemoteFileObject(outdir, jarToSign.getName());
if (outputFile.exists()) fileSS.delete(outputFile, monitor);

//send the command
op = new SimpleCommandOperation(cmdSS, jarToSign.getParentRemoteFile(), true);
op.runCommand("sign " + jarToSign.getAbsolutePath() + " nomail " +
               outdir.getAbsolutePath(), true);

//wait for completion locally
long maxWait = System.currentTimeMillis() + 120000; //max 2 minutes
while (System.currentTimeMillis() < maxWait && !monitor.isCanceled()) {
  outputFile.markStale(true, true);
  outputFile = fileSS.getRemoteFileObject(outputFile.getAbsolutePath());
  if (outputFile.exists() && outputFile.getLength() > jarToSign.getLength()) {
    result = outputFile;
    break;
  }
  Thread.sleep(1000);
}
RSE Tools for Remote Shells and Commands

Model

- IRemoteCmdSubSystem
- IRemoteCommandShell
- IHostShell
- IRemoteOutput
- IRemoteError
- IRemoteShellChangeEvent
- IRemoteShellOutputListener
- IRemoteShellChangeEvent

Subsystem

Service

Controller

- RemoteCommandHelpers
- SimpleCommandOperation
Ex. 4: RemoteCmdSubSystem – Run a Command

- Goal: Register a context menu action that's valid on a remote resource. When executed, run the “nm” command on it and display results in a dialog.

- Tasks:
  - Use PDE, New Plugin, popupMenu Wizard to create action
  - In run() method:
    - Get IRemoteFile for selected resource
    - Use RemoteCommandHelpers to get the proper IRemoteCmdSubSystem
    - Use SimpleCommandOperation to run nm and parse results

- Solution:
  - org.eclipsecon.tmtutorial.nm.ListSymbolsActionDelegate

- Note: exactly the same way you can run commands, and upload/download (like from Ex.3) in LaunchConfigurations as well…
Ex. 4: Remote Command - Solution

```java
private List readOutput(IRemoteFile file) {
    List lines = new ArrayList();
    IRemoteCmdSubSystem cmdSS = RemoteCommandHelpers.getCmdSubSystem(
        file.getParentRemoteFileSubSystem().getHost());
    SimpleCommandOperation op = new SimpleCommandOperation(
        cmdSS, file.getParentRemoteFile(), true);
    String cmdString = "nm " + file.getName();
    try {
        op.runCommand(cmdString, true);
    } catch (Exception e) {}
    String line = op.readLine(true);
    while (line != null) {
        lines.add(line);
        line = op.readLine(true);
    }
    return lines;
}
```
Alternative: Doing it on the Service layer

- Less overhead for events
- See, for instance, LinuxShellProcessService.listAllProcesses()

```java
IShellService shellService = null;
ISubSystem[] subSystems = host.getSubSystems();
for (int i = 0; subSystems != null && i < subSystems.length; i++) {
    if (subSystems[i] instanceof IShellServiceSubSystem) {
        shellService
            = (IShellServiceSubSystem) subSystems[i].getShellService();
        break;
    }
}
if (shellService != null) {
    IHostShell hostShell = shellService.launchShell(
        new NullProgressMonitor(), "", null); //NON-NLS-1$
    hostShell.addOutputListener(new IHostShellOutputListener() {
        public void shellOutputChanged(IHostShellChangeEvent event) {
            IHostOutput[] output = event.getLines();
            System.out.println(output.getString());
        }
    });
    hostShell.writeToShell("ps");
}
```
Ex. 5: Mass Command Execution on many Hosts

- Goal: Create an RSE View which provides an entry field for typing commands. These are sent to a number of previously selected hosts in parallel. Output from running the command is shown in one view per host.

- Tasks:
  - This is an advanced one in terms of writing the UI
  - But the RSE part is simple, but you should know all the concepts by now
  - We’ll just read and inspect the code together

- Solution:
  - org.eclipsecon.tmtutorial.multishell
The Multishell

- Window > Show View > Other > Remote Systems > RSE Multishell
- Commands are sent to any selection of hosts in parallel
- Shell tabs allow to review results
Ex. 5: Mass Command Execution - Solution

```java
public void sendInput(String inputStr) {
    IRemoteCmdSubSystem[] sses = getCmdSubSystems();
    for (int i = 0; i < sses.length; i++) {
        IRemoteCmdSubSystem ss = sses[i];
        IRemoteCommandShell input = getShellFor(ss);
        if (input != null) {
            try {
                ss.sendCommandToShell(new NullProgressMonitor(), inputStr, input);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    _inputEntry.getTextWidget().setText("");
    _inputEntry.getTextWidget().setFocus();
}
```
Wrapping up part I: What you learned

- Ex1: ISystemRegistry – ISystemProfile, IHost, Events
- Ex1a: **Model objects**: Property Sets, IConnectorService
- Ex2: **Model – Adapter layers**, ISystemViewElementAdapter
- Ex3: **Service – Subsystem layers**, IRemoteFileSubSystem
  - SystemRemoteFolderDialog, UniversalFileTransferUtility
  - Ex3a: Doing more with IRemoteFile
- Ex4: **IRemoteCmdSubSystem** SimpleCommandOperation
  - Or on the Service Layer: IHostShellOutputListener
- Ex5: Multishell - A practical example
Part II: Extending RSE

• Up to now, we’ve been building tools that use existing RSE connections and services
• Now we’re going to add new connection types, subsystems and filters
• These examples are part of the standard RSE examples and tutorial, which are available on TM downloads and update site
• We’ll browse through the code and explain the concepts here
Ex. 6: Adding a custom IFileService (FTP)

• Goal: Add a new protocol (FTP) for using the RSE Remote File Browser on it. Works exactly the same for other protocols (want to do WebDAV?)

• Tasks:
  - Have a generic **Service** for FTP (independent of RSE). Write an IFileService wrapper for it, using IHostFile objects as model.
  - Register the **subsystemConfigurations** extension point. Re-use FileServiceSubsystem, but adding the plumbing for an FTP ConnectorService.
  - Write an FTPFileAdapter, and register an AdapterFactory for it in the Activator.

• Solution:
  - org.eclipse.rse.subsystems.files.ftp
<extension point="org.eclipse.rse.ui.subsystemConfiguration">
  <configuration
    systemtypes="Linux;Unix;AIX"
    name="%Files"
    description="%FilesDescription"
    iconlive="icons/full/obj16/systemfileslive_obj.gif"
    icon="icons/full/obj16/systemfiles_obj.gif"
    category="files"
    class="org.eclipse.rse.subsystems.files.ftp.FTPFileSubSystemConfiguration"
    vendor="Eclipse.org"
    id="ftp.files">
  </configuration>
</extension>

Extends FileServiceSubSystemConfiguration

- Just adding the "plumbing" to hook it up with the FTP ConnectorService and FileService
- Plugin will register the Adapters
The new “factory” for FTP Subsystems
Connects the FileServiceSubSystem to a particular instance of an FTPService.
Adapts FTPHostFile to have appropriate UI properties for the RSE Views
Ex. 7: Custom Subsystem with Filters (Developer)

- Goal: Add a new subsystem for completely new kind of resources.

- Tasks:
  - Register the `subsystemConfigurations` extension point. Write your own Subsystem from scratch this time.
  - Write an Adapter for your model objects, with an AdapterFactory, and register it in the Activator.
  - `ISubSystemConfiguration` allows you to configure Filters etc.
  - For the UI part of it, use `ISubsystemConfigurationAdapter`

- Solution:
  - `org.eclipse.rse.examples.tutorial`
A Commercial Implementation (Model, Part IV)

**Model**

- ISubsystemConfiguration
  - Provides special filtering

- ISubsystem
  - ...

- ISystemFilterReference
  - ...

- Original Model Objects
  - from previous generation of product

**View**

- Registry/Connection Filter
  - User-friendly filter editing

- ISubsystemConfigurationAdapter
  - Presentation of Subsystem, Options for Filters

- ISystemViewElementAdapter
  - Adapts them to RSE
TM for Enterprise: IBM WebSphere Developer
Wrapping up part II: What you learned

• Ex6: **Your Own Service** –
  Extension Points `systemTypes`, `subsystemConfigurations`
  - Adding an `IFileService` by registering a new configuration and re-using `IFileServiceSubSystem`
  - Creating an `IConnectorService`
  - Creating an `IHostFileToRemoteFileAdapter`

• Ex7: **Your Own Subsystem**
  - `AdapterFactory`, `ISubsystemConfigurationAdapter`
  - `ISystemViewElementAdapter`
  - `SystemFilterStringEditPane`
TM 3.0 Plans (subset)

• Committed
  ◆ Contribute user actions
  ◆ Import/Export connections and filters to files
  ◆ Improve UI/Non-UI Splitting
  ◆ Improve Lazy Loading and Componentization
  ◆ Add Windows CE Subsystem

• Proposed
  ◆ Cleanup and harden APIs
  ◆ Fix and improve the RSE EFS (Eclipse Filesystem) integration

• See the full plan at
  http://wiki.eclipse.org/DSDP/TM
Upcoming API Changes

• UI / Non-UI Separation:
  - ISubsystemConfiguration, ISubsystem, IConnectorService

• RSE SystemMessage refactoring
  - To be more aligned with standard Eclipse NLS

• But in most cases, 3.0 will be compatible with 2.0
Mission, Goals and Future

- **DSDP Mission**: Create an open, extensible, scalable, and standards-based development platform to address the needs of the device (embedded) software market [...]

- **TM Mission**: Create data models and frameworks to configure and manage remote systems, their connections, and their services.

- **Work in Progress (Technology Sub-Groups)**
  - Component-Based Launching (CBL)
  - Multi-core / Multi-target support through connection groups
  - Adapters for Target access control (shared board labs)

- **Ideas being discussed**
  - Connection Model for HW Debugging (SPIRIT, complex connector setup)
  - Flexible Target Connector framework, Connector plumbing algorithm

- See the TM Wiki, and the TM Use Cases Document
  
Thank You!

• Resources and Pointers
  • TM Homepage, TM Wiki, Newsgroup, Mailinglist
  • > Developer Resources: CVS Team Project Sets, TM Bug Process with many good queries, Committer HOWTO,

• Feel free to contact us at any time... We also have lots of nice „bugday“ bugs
• Questions & Answers

• Join the DSDP & TM BoF!