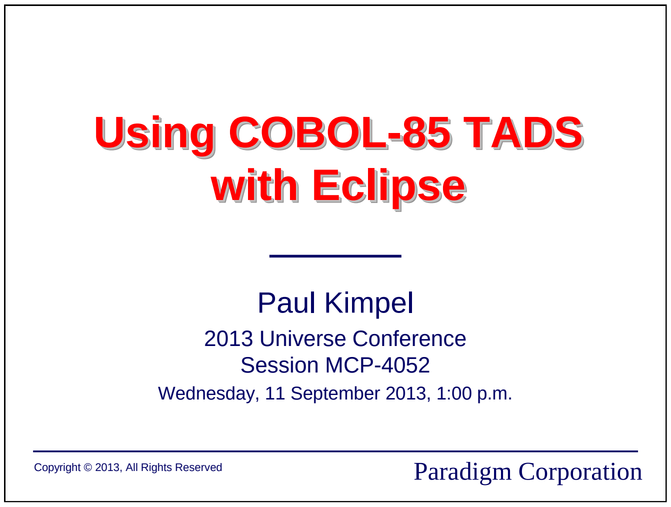

Using COBOL-85 TADS with Eclipse

Paul Kimpel

2013 Universe Conference
Session MCP-4052
Wednesday, 11 September 2013, 1:00 p.m.

Copyright © 2013, All Rights Reserved

Paradigm Corporation

Using COBOL-85 TADS with Eclipse

2013 Universe Conference
Chicago, Illinois

Session MCP-4052

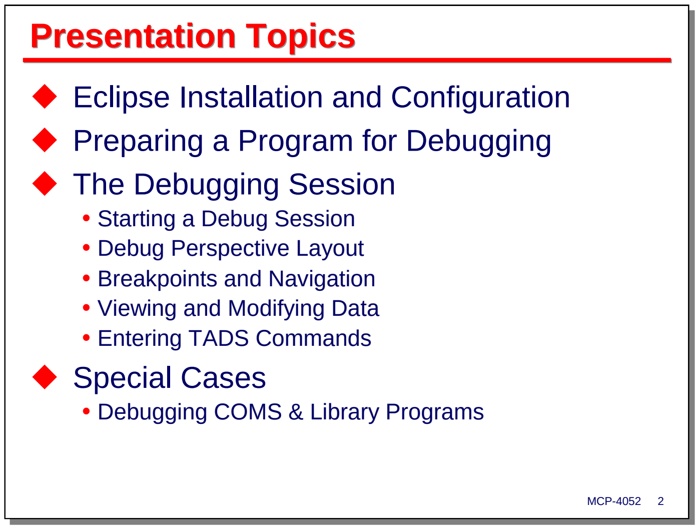
Wednesday, 11 September 2013, 1:00 p.m.

Paul Kimpel

Paradigm Corporation
San Diego, California

http://www.digm.com

e-mail: paul.kimpel@digm.com



Presentation Topics

- Eclipse Installation and Configuration
- Preparing a Program for Debugging
- The Debugging Session
  - Starting a Debug Session
  - Debug Perspective Layout
  - Breakpoints and Navigation
  - Viewing and Modifying Data
  - Entering TADS Commands
- Special Cases
  - Debugging COMS & Library Programs

MCP-4052 2

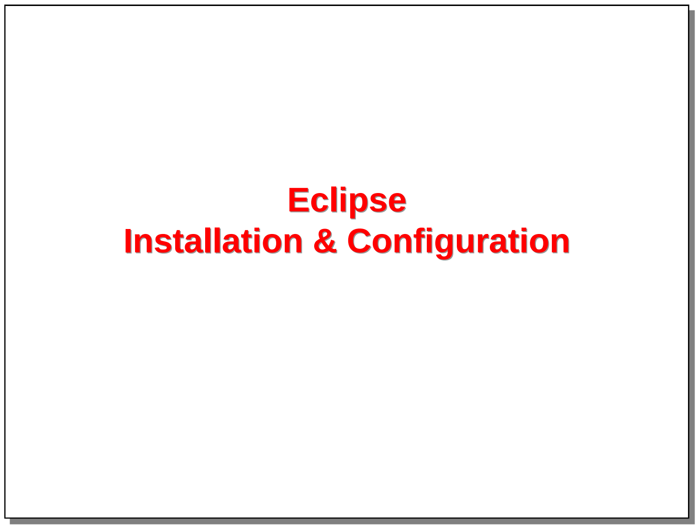
Eclipse is an open-source Integrated Development Environment (IDE), originally developed by IBM for Java. Eclipse is a generic IDE framework, however, and it has been adapted for use with other languages and systems. Unisys has adopted Eclipse for use as and IDE with MCP systems.

The Unisys MCP IDE for Eclipse can be used in many ways – as an Editor, as an application build tool, and as an interface for source code management, among others. In this presentation, we will discuss using the MCP IDE for Eclipse to debug programs, and in particular COBOL-85 programs using a variant of the MCP Test and Debug System, COBOL-85 TADS.

I will discuss briefly how you can acquire Eclipse and the MCP IDE (they're free), and how to install and configure them.

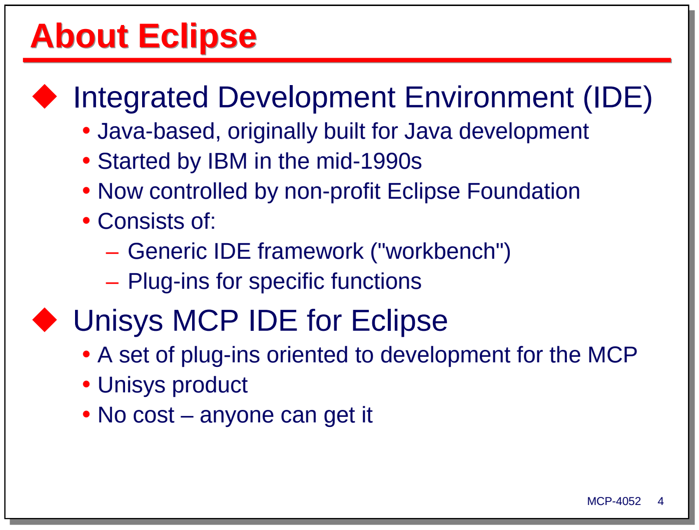
Next, I will talk about how you prepare an MCP-based COBOL-85 program for debugging with the IDE.

The bulk of the presentation will be given over to a discussion and demonstration of the debugging facilities provided by the MCP IDE and how you use them. I'll finish with a brief discussion on how to deal with a couple of special cases, COMS and Library programs.



Eclipse Installation & Configuration

Let us being by talking about how you can acquire Eclipse, install it, and configure it for use.



About Eclipse

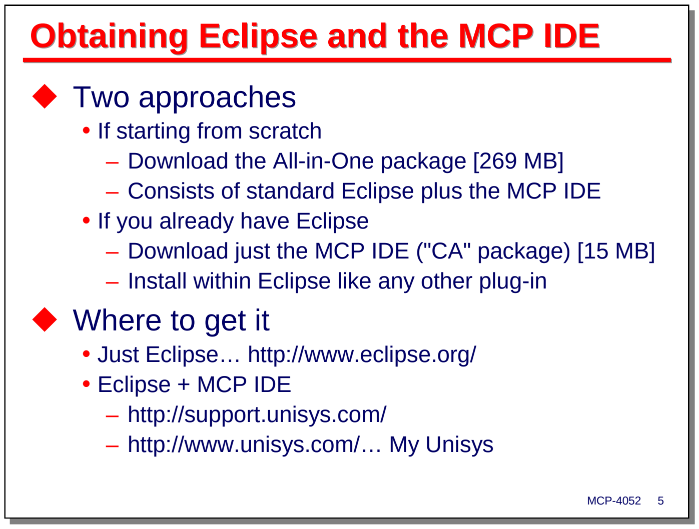
- Integrated Development Environment (IDE)
  - Java-based, originally built for Java development
  - Started by IBM in the mid-1990s
  - Now controlled by non-profit Eclipse Foundation
  - Consists of:
    - Generic IDE framework ("workbench")
    - Plug-ins for specific functions
- Unisys MCP IDE for Eclipse
  - A set of plug-ins oriented to development for the MCP
  - Unisys product
  - No cost – anyone can get it

MCP-4052 4

Eclipse is an Integrated Development Environment, or IDE. It is written almost entirely in Java, and was originally created to support Java development. It was developed by IBM's Object Technologies International (OTI) subsidiary, and is an outgrowth of their earlier Visual Age for Java product. Eventually the product became open source, and the not-for-profit Eclipse Foundation was formed to act as a steward for the code.

Rather than just building an IDE for Java, however, OTI structured Eclipse as a framework that could be modified and adapted for other purposes. Eclipse consists of a generic IDE core, called the "workbench." With this core are a series of plug-in modules that add specific functionality to the workbench. Initially, the plug-ins consisted of the Java Development Toolkit (JDT) and the Plug-in Development Environment (PDE), which allows you to use Eclipse to create new plug-ins. A large number of plug-ins have been developed over the years to adapt Eclipse to different purposes, different languages, and different systems.

Unisys has developed a series of plug-ins oriented to development for MCP applications. The collection of those plug-ins is referred to at the MCP IDE for Eclipse. This is a proprietary Unisys product, but it is publicly available and can be downloaded by anyone at no cost.



Obtaining Eclipse and the MCP IDE

- Two approaches
  - If starting from scratch
    - Download the All-in-One package [269 MB]
    - Consists of standard Eclipse plus the MCP IDE
  - If you already have Eclipse
    - Download just the MCP IDE ("CA" package) [15 MB]
    - Install within Eclipse like any other plug-in
- Where to get it
  - Just Eclipse… http://www.eclipse.org/
  - Eclipse + MCP IDE
    - http://support.unisys.com/
    - http://www.unisys.com/… My Unisys

MCP-4052 5

There are two ways to obtain Eclipse and the MCP IDE.

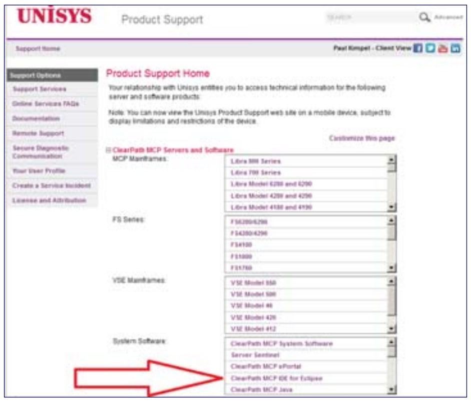
First, if you do not currently have Eclipse and are starting from scratch, it is best to download what is called the All-in-One package. This consists of a current release of the standard Eclipse IDE for Java EE Developers, plus the Unisys IDE plug-ins, all integrated and ready to use. This package is a 369 MB download.

Second, if you already have Eclipse and merely want to add the MCP IDE to it, you can obtain just that portion as a 15 MB download called the "CA" (for Composite Application) package. You then install this package within Eclipse just like any other plug-in.

Where can you get it? To get just Eclipse itself, you can go to the Eclipse web site at http://www.eclipse.org/. To get the Unisys All-in-One or CA packages, you have two choices:

- If you have access to the Unisys support site, http://support.unisys.com/, you can download both packages from there, as we will see on the next slides.
- If you do not have access to the support site, you can download the MCP IDE from the public Unisys web site, http://www.unisys.com, in the My Unisys section of the site. You must register for a My Unisys account, but it is free, and anyone can do it.

From Unisys Support – Step 1



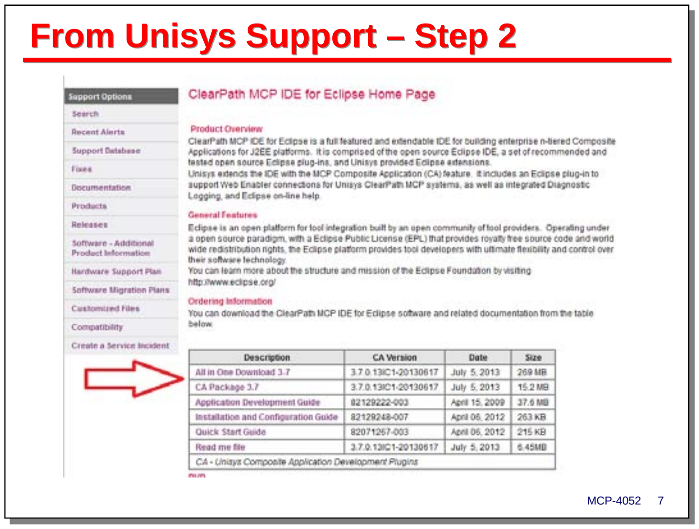
Product Support Home

Your relationship with Unisys entitles you to access technical information for the following server and software products:

Note: You can now view the Unisys Product Support web site on a mobile device, subject to display limitations and restrictions of the device.

MCP-4052 6

After logging in to the Unisys support site, you should be on the Support Home page. Near the bottom should be a link for the "ClearPath MCP IDE for Eclipse." Click that link.



From Unisys Support – Step 2

Support Options
- Search
- Recent Alerts
- Support Database
- Fixes
- Documentation
- Products
- Releases
- Software - Additional Product Information
- Hardware Support Plan
- Software Migration Plans
- Customized Files
- Compatibility
- Create a Service Incident

ClearPath MCP IDE for Eclipse Home Page

Product Overview

ClearPath MCP IDE for Eclipse is a full featured and extendable IDE for building enterprise n-tiered Composite Applications for J2EE platforms. It is comprised of the open source Eclipse IDE, a set of recommended and tested open source Eclipse plug-ins, and Unisys provided Eclipse extensions.
Unisys extends the IDE with the MCP Composite Application (CA) feature. It includes an Eclipse plug-in to support Web Enabler connections for Unisys ClearPath MCP systems, as well as integrated Diagnostic Logging, and Eclipse on-line help.

General Features

Eclipse is an open platform for tool integration built by an open community of tool providers. Operating under a open source paradigm, with a Eclipse Public License (EPL) that provides royalty free source code and world wide redistribution rights, the Eclipse platform provides tool developers with ultimate flexibility and control over their software technology.
You can learn more about the structure and mission of the Eclipse Foundation by visiting http://www.eclipse.org/

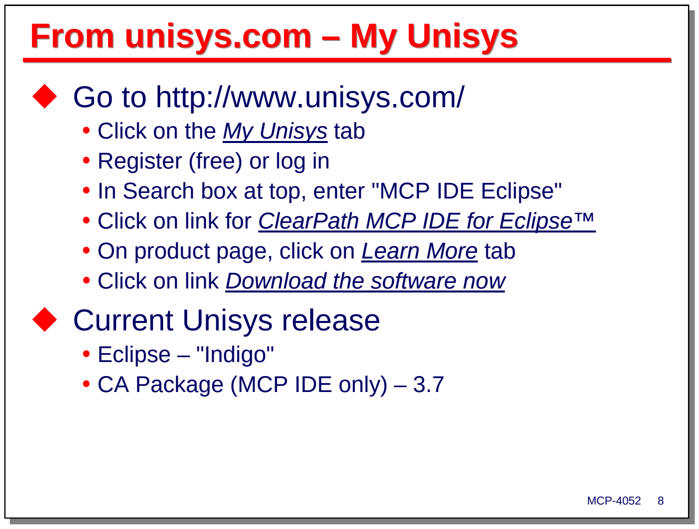
Ordering Information

You can download the ClearPath MCP IDE for Eclipse software and related documentation from the table below.

Description	CA Version	Date	Size
All in One Download 3.7	3.7.0.13IC1-20130617	July 5, 2013	269 MB
CA Package 3.7	3.7.0.13IC1-20130617	July 5, 2013	15.2 MB
Application Development Guide	82129222-003	April 15, 2009	37.6 MB
Installation and Configuration Guide	82129248-007	April 06, 2012	263 KB
Quick Start Guide	82071267-003	April 06, 2012	215 KB
Read me file	3.7.0.13IC1-20130617	July 5, 2013	6.45MB
CA - Unisys Composite Application Development Plugins			

MCP-4052 7

The the page that results from the link on the prior slide shows a number of resources for the MCP IDE. You can select the All-in-One or CA package at the top of the list, as well as a few documents relating to the MCP IDE and Composite Application package.



From unisys.com – My Unisys

- Go to http://www.unisys.com/
  - Click on the *My Unisys* tab
  - Register (free) or log in
  - In Search box at top, enter "MCP IDE Eclipse"
  - Click on link for *ClearPath MCP IDE for Eclipse™*
  - On product page, click on *Learn More* tab
  - Click on link *Download the software now*
- Current Unisys release
  - Eclipse – "Indigo"
  - CA Package (MCP IDE only) – 3.7

MCP-4052 8

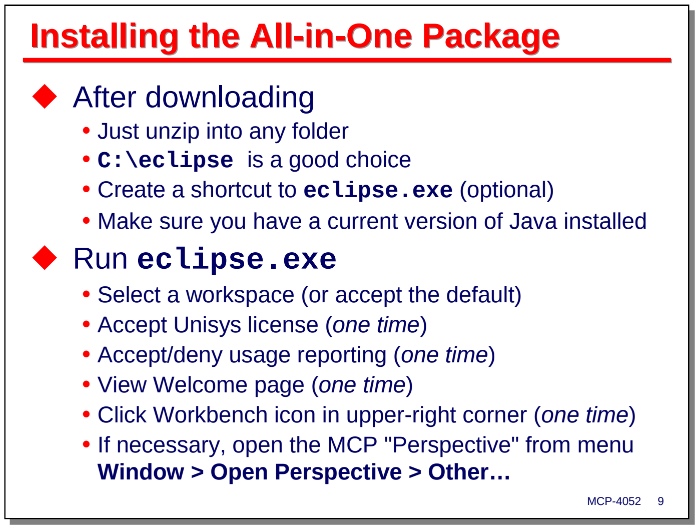
To acquire the MCP IDE without going to the support site, go to the public Unisys web site, http://www.unisys.com. At the top of the home page, click the "My Unisys" tab. If you have already registered for this, log in, otherwise create a new account.

Once you are logged in, use the Search box at the top of the page to find entries for "MCP IDE Eclipse". From the resulting page of hits, select the first one for "ClearPath MCP IDE for Eclipse". On the resulting page, click the tab for "Learn More." On that tab should be a link for "Download the software now." Click that link.

From the resulting page you can accept the Unisys license agreement and download the All-in-One or CA packages.

As this presentation is being prepared, the current versions of the software available from Unisys are:

- For Eclipse itself, "Indigo."
- For the Unisys CA package containing the MCP IDE, 3.7. This works with at least MCP 12.0 through 15.0, and may work with earlier versions as well.



Installing the All-in-One Package

- After downloading
  - Just unzip into any folder
  - **`C:\eclipse`** is a good choice
  - Create a shortcut to **`eclipse.exe`** (optional)
  - Make sure you have a current version of Java installed
- Run **`eclipse.exe`**
  - Select a workspace (or accept the default)
  - Accept Unisys license (*one time*)
  - Accept/deny usage reporting (*one time*)
  - View Welcome page (*one time*)
  - Click Workbench icon in upper-right corner (*one time*)
  - If necessary, open the MCP "Perspective" from menu **Window > Open Perspective > Other…**

MCP-4052 9

Installing the All-in-One package is really easy. All you need to do is unzip it into a folder on your workstation. In this presentation, we will assume you are using Microsoft Windows, but with minor differences, these instructions should work on UNIX, Linux, and Apple Macintosh systems as well.

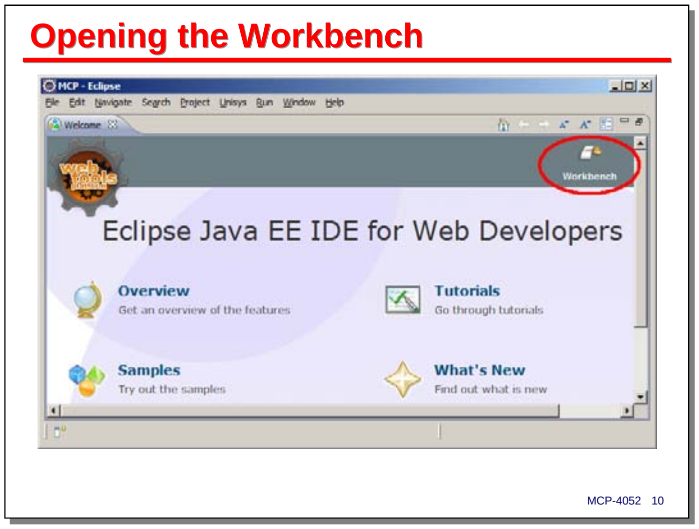
A typical approach is to unzip the All-in-One package download to **`C:\eclipse`**. On Windows, there is a small startup program, **`eclipse.exe`**. You may wish to create a shortcut to that program and place it on your desktop or Start menu.

Recall that Eclipse and its plug-ins are written in Java, so make sure you have a current version of at least the Java Runtime Environment (JRE) installed on your workstation before proceeding.

To initiate Eclipse and the IDE on a Windows system, simply execute **`eclipse.exe`**.

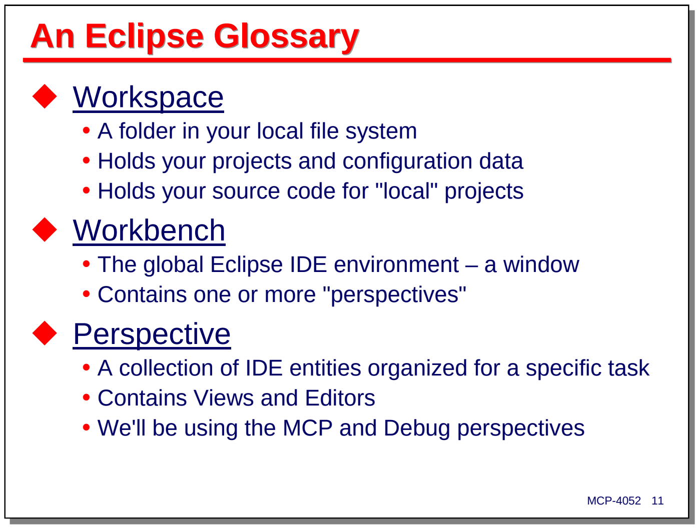
- The first thing Eclipse does is prompt you for your "workspace." This is a directory in your local file system where Eclipse will store your projects and configuration data. You can have multiple workspaces, and each workspace can support multiple projects. Choose whatever directory suits you. There is an option to make this the default workspace. If you do not select that option, Eclipse will prompt you for a workspace each time you initiate it.
- The first time you run the All-in-One package there are some one-time events that occur:
  – The MCP IDE will prompt you to accept the Unisys license.
  – Eclipse will request that you either accept or deny permission for it to do usage reporting. Accepting this will allow Eclipse to transmit anonymous usage statistics for analysis by various Eclipse development teams.
  – Eclipse will display its Welcome page. There are links here for tutorials and other background information. You can view these or not as you prefer. There is a menu selection in the workbench that will allow you to get back to this page later.
  – To get started with Eclipse and the MCP IDE, you need to click the "Workbench" icon in the upper-right corner of the page, as shown on the next slide. Subsequent runs of the IDE should bypass the welcome page and open directly in the Workbench.

If you downloaded the All-in-One package from Unisys, clicking the Workbench icon should open the MCP "Perspective" at this point. If it does not, select **Window** > **Open Perspective** > **Other…** from the menus.



This slide shows the Eclipse Welcome page and the location of the Workbench icon. Click this icon to enter the Workbench user interface and enable the MCP IDE.

You can return to the Welcome page later by selecting **Help > Welcome** from the menus.



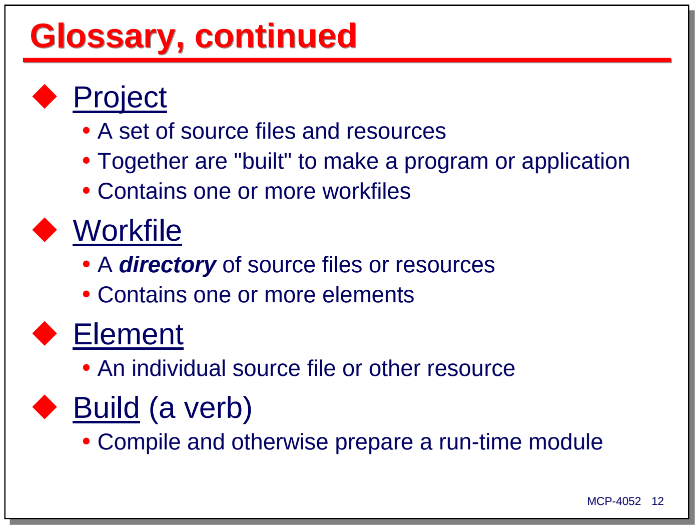
An Eclipse Glossary

- Workspace
  - A folder in your local file system
  - Holds your projects and configuration data
  - Holds your source code for "local" projects
- Workbench
  - The global Eclipse IDE environment – a window
  - Contains one or more "perspectives"
- Perspective
  - A collection of IDE entities organized for a specific task
  - Contains Views and Editors
  - We'll be using the MCP and Debug perspectives

MCP-4052 11

Eclipse has some *very* peculiar terms for the entities you work with. When you first approach it from an MCP background, these terms can be downright misleading. In an effort to help you overcome this part of the learning curve, the next two slides present a glossary.

- **Workspace** – as mentioned previously, this is simply a folder in your local file system, or perhaps on a file server to which you have access. This folder is used to store your Eclipse projects and configuration data. Traditionally the workspace holds your source code, and for so-called "local" MCP projects, that is still true. For the purpose of this presentation, we will be dealing with "remote" MCP projects, where the source code is resident on the MCP host.
- **Workbench** – this is the global Eclipse environment. Basically, it's a window managed by the Eclipse framework. The various plug-ins display their user interfaces within the Workbench. Typically, these user interfaces are in the form of "perspectives."
- **Perspective** – this is a collection of IDE entities that are organized and arranged within the Workbench for a specific type of task. The MCP IDE is one such perspective, and one of the two we will be discussing most during this presentation. The other is the Debug Perspective. Primarily, perspectives consist of user interface entities known as Views and Editors.
  - A **View** is just a presentation of data or state within the IDE. You can open and close Views at will, and move them around within the Workbench.
  - An **Editor** is a special type of View oriented to creating and maintaining some type of resource. The most common example is for source code. There are specialized Editors for different languages.



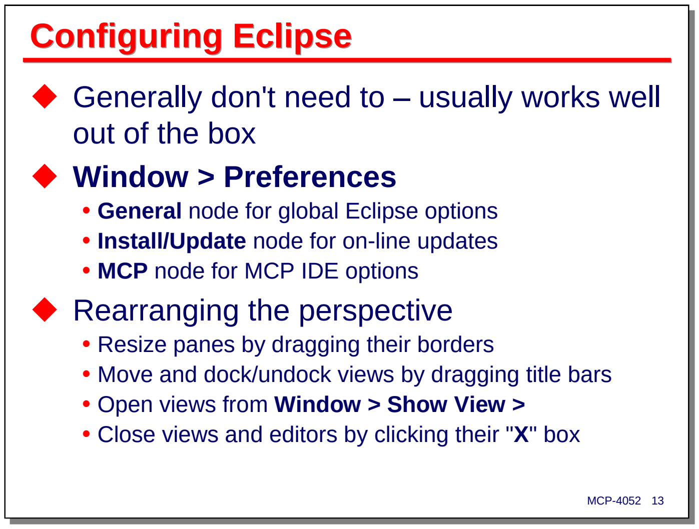
Glossary, continued

- Project
  - A set of source files and resources
  - Together are "built" to make a program or application
  - Contains one or more workfiles
- Workfile
  - A ***directory*** of source files or resources
  - Contains one or more elements
- Element
  - An individual source file or other resource
- Build (a verb)
  - Compile and otherwise prepare a run-time module

MCP-4052 12

Continuing with the brief Eclipse glossary…

- **Project** – this is a unit of development, and its concept is similar to that for projects in other development environments. A project typically consists of a set of source files, the object code compiled ("built") from those source files, plus other resources necessary to create an application or a component of an application. A project generally contains one or more "workfiles." In this presentation, we are going to keep it simple, and our projects will contain only a single MCP source file. For many MCP development tasks that's all you need, but more complex project structures are possible.
- **Workfile** – of all the Eclipse terms, this one is the most baffling and misleading. It is not a file at all, but rather a *directory* or folder of files in some file system. For MCP projects, that directory can be on your local workstation or on the MCP host, although in this presentation we will concentrate on the latter. A workfile contains one or more "elements."
- **Element** – this is perhaps the second most baffling of the mysterious Eclipse terms. For our purposes, it simply means *file*, and in particular, a source file.
- **Build** – this is a verb, and its usage is similar to that in other development environments, but it is a little foreign to typical MCP development. In general, it means to assemble and prepare the source files and other resources of a project to generate a run-time module. For our purposes, it simply means "compile."



Configuring Eclipse

- Generally don't need to – usually works well out of the box
- **Window > Preferences**
  - **General** node for global Eclipse options
  - **Install/Update** node for on-line updates
  - **MCP** node for MCP IDE options
- Rearranging the perspective
  - Resize panes by dragging their borders
  - Move and dock/undock views by dragging title bars
  - Open views from **Window > Show View >**
  - Close views and editors by clicking their "**X**" box

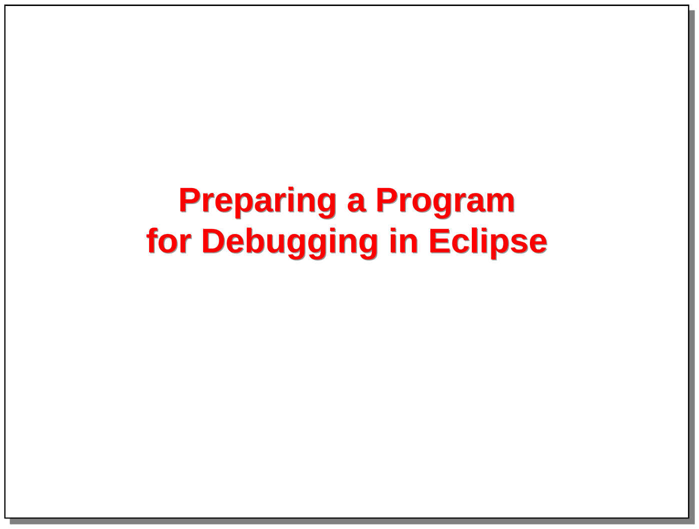
MCP-4052 13

Eclipse has a large set of options you can tweak to change the way it works. Most of these do not apply to use with the MCP IDE. Generally, you don't need to change a thing, and it works well out of the box.

If you want to examine the Eclipse options, choose **Window** > **Preferences** from the menus. There are three nodes in the tree of options to which you may want to pay attention:

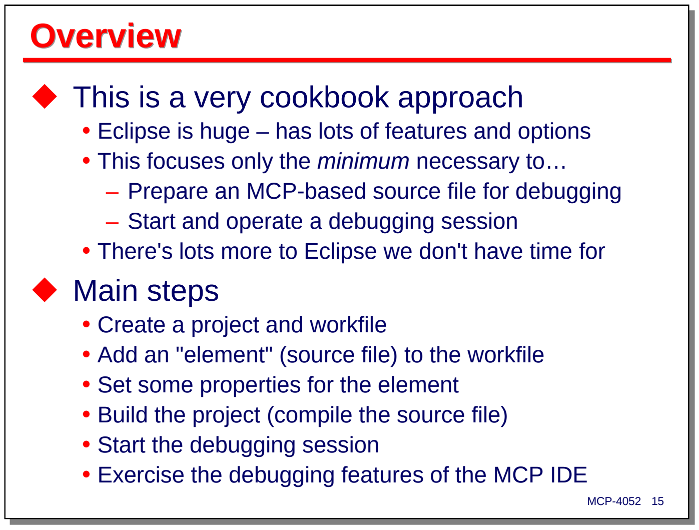
- **General** – this node covers the global Eclipse workbench options
- **Install/Update** – this node allows you to selectively update the various plug-ins and workbench components.
- **MCP** – this node covers the options that apply specifically to the MCP IDE

Another form of configuration is to rearrange the entities of a perspective. Views can be dragged by their title bars, and docked and undocked along the borders of the perspective. You can open new views by selecting **Window** > **Show View** from the menus and picking items from the sub-menu. Note that some MCP IDE views are in the **Other…** item. You can close views simply by clicking the "**X**" in their title bar or tab.



Preparing a Program for Debugging in Eclipse

With that background on installing and configuring Eclipse and the MCP IDE, the next topic is preparing an MCP program for debugging.



Overview

- This is a very cookbook approach
  - Eclipse is huge – has lots of features and options
  - This focuses only the *minimum* necessary to…
    - Prepare an MCP-based source file for debugging
    - Start and operate a debugging session
  - There's lots more to Eclipse we don't have time for
- Main steps
  - Create a project and workfile
  - Add an "element" (source file) to the workfile
  - Set some properties for the element
  - Build the project (compile the source file)
  - Start the debugging session
  - Exercise the debugging features of the MCP IDE

MCP-4052 15

Before we begin this section, note that what follows is a very cookbook approach. Eclipse is huge. It has lots of features and options. There are other ways to do what we are going to cover in the next slides, and there are lots of features and alternatives that are beyond the scope of this presentation.

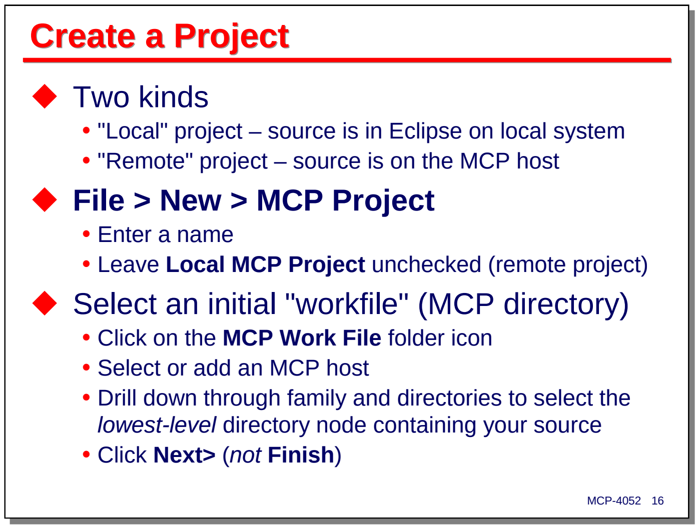
In this presentation, we are going to focus only on the minimum necessary to get you started using Eclipse as a debugging tool. That includes preparing an MCP-based source program for debugging, and starting and operating a debugging session.

The main steps in this section will be covered under the following outline:

- Create a "remote" MCP project and an associated workfile. You can't do much of anything within the Workbench without a project.
- Add an "element" (a source file, in this case) to the workfile.
- Next, we need to set some properties on the element in preparation for building the project. One of those properties will enable the program to be compiled for TADS.
- Build the project – that is, compile the source file to obtain a TADS-enabled object file.
- Start the debugging session within the MCP IDE for Eclipse.
- Exercise the debugging features of the MCP IDE.

One thing to keep in mind while working with the MCP IDE for Eclipse is that Eclipse was originally developed with the idea that the source code, object code, project building, testing, etc., all take place on your local workstation. That isn't the case with MCP projects. Programs must be compiled on an MCP host, and of course, they must run there as well.

The MCP IDE tries to blur the lines between what needs to happen locally and what needs to happen on the MCP host, and generally it does a good job of that, but it must work somewhat outside the original concept of Eclipse in order to do so. Just be aware that you are working with two systems, not one, and that in many cases the MCP IDE is acting as a proxy for things that really take place on the MCP host.



Create a Project

- Two kinds
  - "Local" project – source is in Eclipse on local system
  - "Remote" project – source is on the MCP host
- **File > New > MCP Project**
  - Enter a name
  - Leave **Local MCP Project** unchecked (remote project)
- Select an initial "workfile" (MCP directory)
  - Click on the **MCP Work File** folder icon
  - Select or add an MCP host
  - Drill down through family and directories to select the *lowest-level* directory node containing your source
  - Click **Next>** (*not* **Finish**)

MCP-4052 16

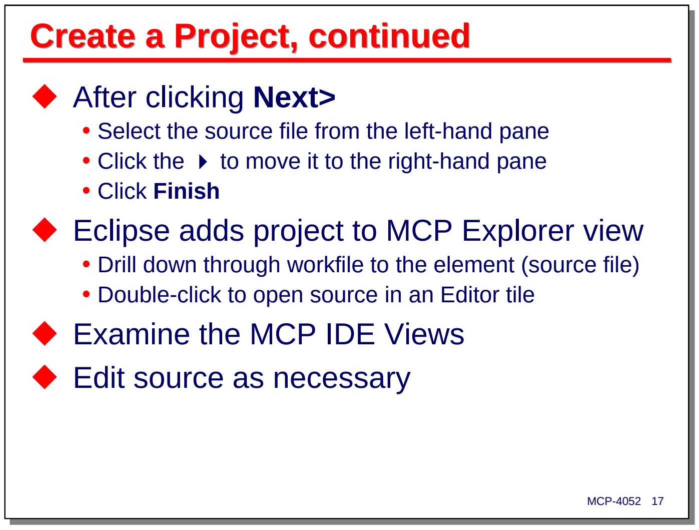
The first step in preparing to debug a program with the MCP IDE is to create a project. With respect to the MCP IDE, there are two kinds of projects:

- Local projects maintain their source code within Eclipse on your workstation. The source files are transferred by Eclipse as necessary to the MCP host to compile and otherwise build your project.
- Remote projects work in the more traditional sense, in that the source code is stored on the MCP host. Portions of the source are transferred to Eclipse as necessary for viewing, editing, etc.

In this presentation, we are only going to use a remote project.

To create that project, select **File** > **New** > **MCP Project** from the menus. This will open a dialog box to set the initial properties for the project.

- Enter a name for the project. This name becomes the name of a folder in your Eclipse workspace, so it must obey the file-naming conventions of your local workstation.
- Leave the **Local MCP Project** checkbox *unchecked*. This is what makes it a "remote" project.
- Next, select an initial workfile for the project. This is a directory in the MCP file system that contains your source file. It is possible to enter the specification for the directory manually into the workfile textbox, but the syntax is a little arcane. The easier thing to do is to click the folder button to the right of that textbox.
  - When you click that folder button, the IDE will open a dialog to select an MCP directory. It is a typical two-pane explorer dialog. In the left pane is a tree of MCP servers and directories. The right pane will show details for whatever is currently selected in the left pane.
  - If you are just starting out with the IDE, you probably don't have any MCP servers defined, so the first thing is to set one up. Click the **Add Host** (**+**) button along the top of the two panes. this will open another dialog box where you can specify the MCP host name or IP address, an MCP usercode and password, plus other other log-on options such as chargecode, Kerberos authentication, etc.
  - When finished, click the **OK** button, which will save the information you have entered, close that dialog box, and attempt to connect to the specified MCP host. The host should appear in the left-pane tree. It will still be there the next time you run Eclipse. Assuming that the connection to the MCP host is successful, the IDE will display the root nodes (disk families) for that server.
  - What has worked best for me at this point is to drill down through the families and directories to the *lowest-level directory* containing the source file you want to use. Click the **Open** button to select that directory and close the dialog box. Note that we have not yet selected a source file, just the directory that contains it. The source file is selected on the next slide.
  - Back on the New MCP Project dialog, the arcane directory specification will be in the workfile text box. Click the **Next>** button to continue. *Do not click* **Finish** at this point – that will create a confusing situation that you don't want to have.



Create a Project, continued

- After clicking **Next>**
  - Select the source file from the left-hand pane
  - Click the ▸ to move it to the right-hand pane
  - Click **Finish**
- Eclipse adds project to MCP Explorer view
  - Drill down through workfile to the element (source file)
  - Double-click to open source in an Editor tile
- Examine the MCP IDE Views
- Edit source as necessary

MCP-4052 17

After clicking the **Next>** button on the New MCP Project dialog, the IDE will display a two-pane file selection dialog. The names of the files in the selected directory will appear in the left-hand pane. Click the one you want to use, then click the right-pointing arrowhead to move it to the right-hand pane. This adds that file to the project.

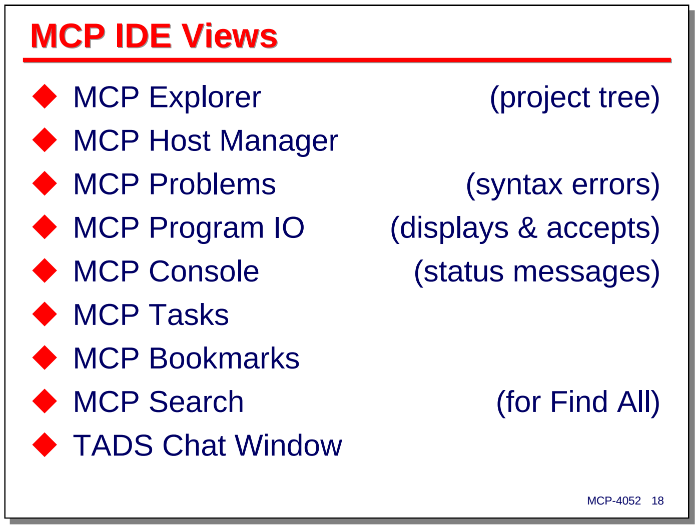
(Note: you can select multiple files in this dialog and add them to the project, but that is not something you typically will want to do in an MCP project for traditional MCP-based applications).

Now you can click the **Finish** button. This will close the New MCP Project dialog. Back on the Workbench, you will see that the project has been added to the MCP Explorer view. Drill into the project, down through the workfile, to display the source file ("element") node.

Double-click the element node (or right-click and select Open from the context menu). This will open an Editor tile in the MCP Perspective and display the text of the MCP source file.

Now that we have a project and a source file loaded in the IDE, this is a good time to examine some of the views and other features of the MCP IDE. We'll do that on the next slide.

The tile on the perspective that just opened is called an Editor for a good reason. This is a text editor with a fairly full set of features, somewhat similar to PWB. You can scroll, view, search, and edit the source code in this tile and save it back to the MCP host. The Editor tile is an interesting subject, but outside the scope of this presentation.



MCP IDE Views

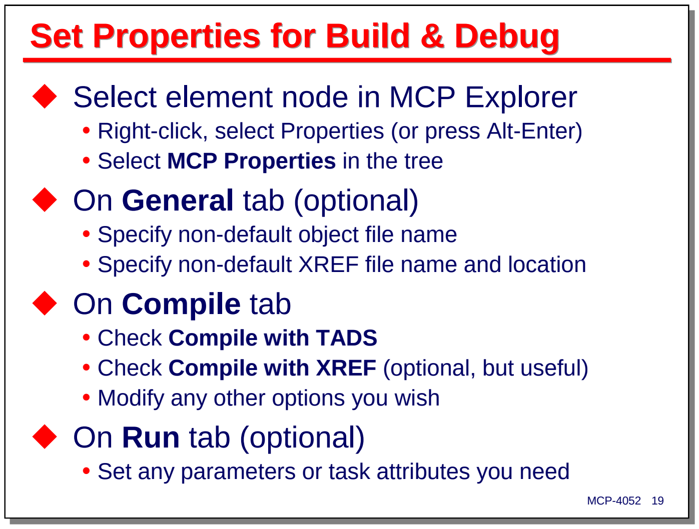
◆ MCP Explorer	(project tree)
◆ MCP Host Manager	
◆ MCP Problems	(syntax errors)
◆ MCP Program IO	(displays & accepts)
◆ MCP Console	(status messages)
◆ MCP Tasks	
◆ MCP Bookmarks	
◆ MCP Search	(for Find All)
◆ TADS Chat Window	

MCP-4052 18

There are a number of Views supported by the MCP IDE. As previously mentioned, you can move these around in the Workbench, and open and close them at will. The main Views for the MCP Perspective are:

- **MCP Explorer** – this lists the projects currently in your workspace. Each project is a tree. You can drill into the tree to expose the workfiles and elements associated with the project. Right-clicking on the nodes will bring up context menus through which you can configure and operate on the nodes.
- **MCP Host Manager** – this lists all of the MCP host configurations and allows you to edit, add, or delete them. This data is not stored in your workspace; it is global to Eclipse (on Windows, it's stored in **`AppData`**).
- **MCP Problems** – this view is used primarily to report syntax errors from compilations.
- **MCP Program IO** – this is used to display messages from MCP-based tasks. It also allows you to respond to **`ACCEPT`** messages.
- **MCP Console** – this will display status and messages from the IDE's interface to the MCP host (the **`NXEDITSUPPORT`** library).
- **MCP Tasks** – this is a developer's to-do list that allows you to record and prioritize development tasks, associate them with resources, and to mark them completed.
- **MCP Bookmarks** – this lists any bookmarks you have placed on lines in your source file. You can create bookmarks with **Edit** > **Add Bookmark…** You can navigate to a bookmarked line by right-clicking on the bookmark in this view and selecting **Go to**.
- **MCP Search** – this lists the lines that were matched by an **Edit** > **Find Text…** command after the **Find All** button was clicked.
- **TADS Chat Window** – this is actually a debugging view, but it also displays in the MCP Perspective by default.

There are additional MCP Views associated with the Debug View, which we will cover later in the presentation.



Set Properties for Build & Debug

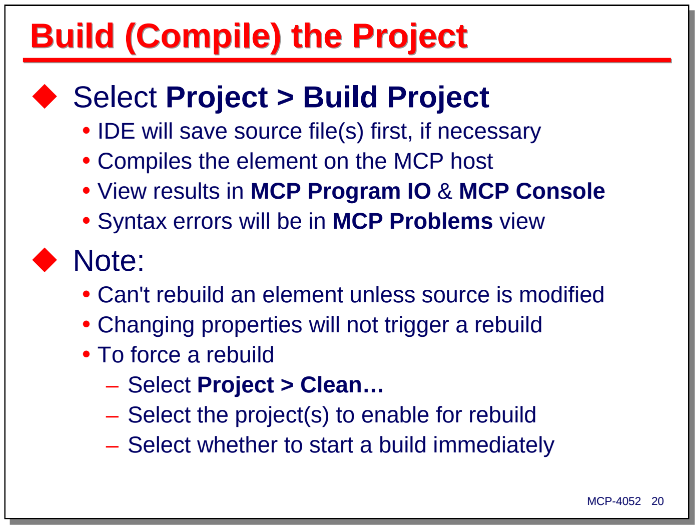
- Select element node in MCP Explorer
  - Right-click, select Properties (or press Alt-Enter)
  - Select **MCP Properties** in the tree
- On **General** tab (optional)
  - Specify non-default object file name
  - Specify non-default XREF file name and location
- On **Compile** tab
  - Check **Compile with TADS**
  - Check **Compile with XREF** (optional, but useful)
  - Modify any other options you wish
- On **Run** tab (optional)
  - Set any parameters or task attributes you need

MCP-4052 19

The next step in preparing the program for a debugging session is to set some properties on the source file, or element, node.

- Right-click on the element node and select **Properties** from the context menu. Alternatively, select the node and press Alt-Enter on the keyboard. Then click the **MCP Properties** node in the tree.
- There are two optional things you can do on the **General** tab:
  - You may specify the name of the object file that will be associated with the source file. If you have already compiled the program on the MCP host, you may need to specify its name here, or perhaps you have a convention for naming object files. The default object name is the CANDE convention of prefixing the source file name by **`OBJECT/`**.
  - You may specify a custom name and location for XREF files. This is only necessary if you are using the XREF feature and you are not using the default file and family names.
- If you are going to be compiling (building) the program in Eclipse, do the following on the **Compile** tab. If you have compiled the program externally and only wish to debug it, you can skip the settings on this tab.
  - Uncheck **Use Default Properties**.
  - Check the **Compile with TADS** box.
  - If you wish to use the XREF feature, check the **Compile with XREF** box. This is optional, but it can be useful.
  - Select any of the other compiler options you wish.
- If your program requires parameters or task attribute settings (including file attribute assignments), specify these on the **Run** tab.
  - For debugging normal programs, you should leave the **Run Object File Directly** radio button selected.
  - For certain special cases, such as debugging library programs, you may need to run a client program to activate the module you are debugging. The **Run Client to Initiate Library** selection is useful for this.
  - If your program requires a lot of complex parameter or task/file attribute setup, you may wish to place all of that in a WFL job and initiate the program that way. In that case, select the **Run using WFL Job** radio button.

In these latter two cases, see the discussion at the end of the presentation on Special Cases and the use of the Debug Listen option to initiate debugging sessions.



Build (Compile) the Project

- Select **Project > Build Project**
  - IDE will save source file(s) first, if necessary
  - Compiles the element on the MCP host
  - View results in **MCP Program IO** & **MCP Console**
  - Syntax errors will be in **MCP Problems** view
- Note:
  - Can't rebuild an element unless source is modified
  - Changing properties will not trigger a rebuild
  - To force a rebuild
    - Select **Project > Clean…**
    - Select the project(s) to enable for rebuild
    - Select whether to start a build immediately

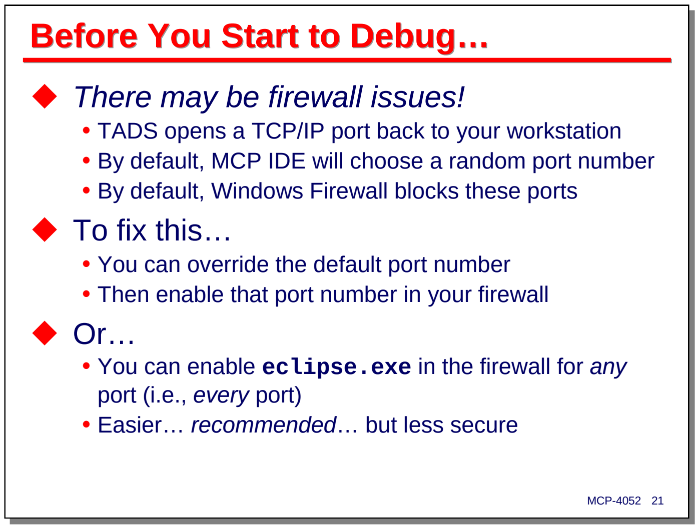
MCP-4052 20

Having created our project and specified any necessary properties, we are now ready to build our project – that is, compile the program. If you have compiled the program externally, you can skip the information on this slide. Otherwise, select **Project > Build Project** from the menus.

- If you have modified the source files, the IDE will save them first.
- The IDE then initiates a compile on the MCP host.
- Messages and status of the compile task will be displayed in the **MCP Program IO** and **MCP Console** views.
- Syntax errors will be displayed in the **MCP Problems** view.

One thing to note is that Eclipse will not rebuild something that it has already built and does not consider to have been modified. In particular, merely changing properties on the source file element will not trigger a rebuild – selecting **Project > Build Project** after that will not do anything. This can be an issue if, say, you forget to check the **Compile with TADS** box on the properties and do not realize it until after you have done a build.

The solution is to force a rebuild by selecting **Project > Clean…** from the menus. This displays a dialog where you can select the projects for which you want to clear the build status. You can also specify on this dialog whether you want to rebuild the project(s) immediately (which normally you will).



Before You Start to Debug…

- *There may be firewall issues!*
  - TADS opens a TCP/IP port back to your workstation
  - By default, MCP IDE will choose a random port number
  - By default, Windows Firewall blocks these ports
- To fix this…
  - You can override the default port number
  - Then enable that port number in your firewall
- Or…
  - You can enable **`eclipse.exe`** in the firewall for *any* port (i.e., *every* port)
  - Easier… *recommended*… but less secure

MCP-4052 21

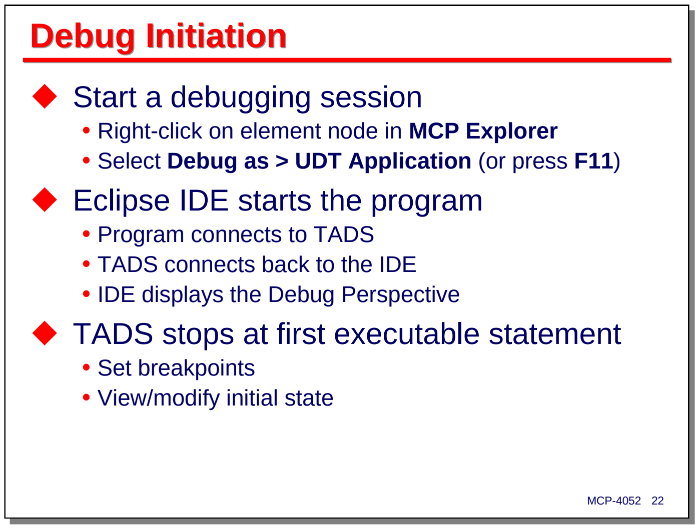
At this point we have prepared our project and have the source and object files ready to use in a debugging session. Before we start one, however, there may be some firewall issues to be addressed.

- TADS is implemented as a library program. When you initiate a program in debugging mode, your program implicitly calls the TADS library. The TADS library then opens a connection back to the client that will control the debugging session.
- When using TADS through the MCP IDE for Eclipse (or PWB, for that matter), TADS opens a TCP/IP port back to Eclipse running on your workstation. By default, TADS chooses a random port number to do this, and by default for most users, Windows Firewall will block that port.

To fix this problem, there are two things you can do on your workstation:

- Override the IDE-chosen port number in the project configuration with a specific port number (we'll see how to do this in a minute). Then open that port number for external access in your Windows Firewall configuration.
- Allow access to the eclipse.exe program for *any* port in your Windows Firewall configuration. On most workstations, Windows will prompt you for this approach the first time you try to debug through Eclipse. This approach is much easier than the one above, and I actually recommend doing this, but it is less secure.

Of course, you can always just disable Windows Firewall in toto, but that is generally not a good idea.



Debug Initiation

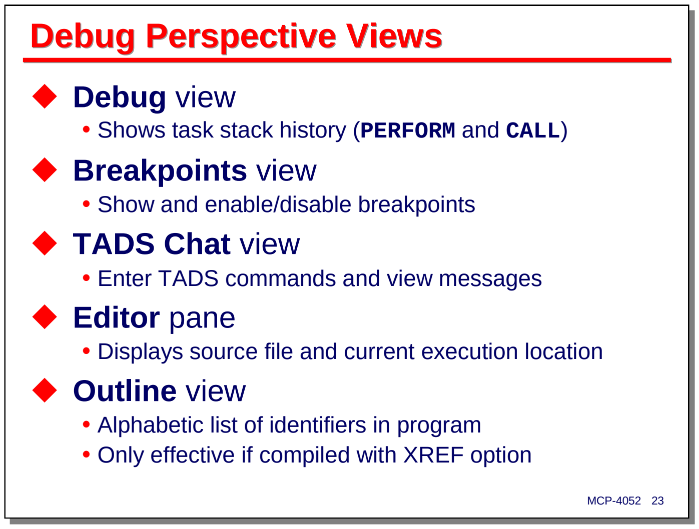
- Start a debugging session
  - Right-click on element node in **MCP Explorer**
  - Select **Debug as > UDT Application** (or press **F11**)
- Eclipse IDE starts the program
  - Program connects to TADS
  - TADS connects back to the IDE
  - IDE displays the Debug Perspective
- TADS stops at first executable statement
  - Set breakpoints
  - View/modify initial state

MCP-4052 22

This slide discusses how to override the randomly-chosen TADS port number in the MCP IDE so that you can open a specific port in your local workstation's firewall.

Setting up the debugging configuration for the IDE is somewhat complex, and difficult to get right when you do it manually. I have found that the easiest thing to do is just to start a debugging session and let it fail with a firewall blockage. That will set up a debugging configuration, which you can then proceed to update with your chosen port number.

- First, start a debugging session. To do that, right-click on the program element in the We need them. view and select **Debug as > UDT Application**. Alternatively, just select the element node and press F11.
- The Debug Perspective will display, but it will probably not be able to initialize properly, and will hang. Select **Run** > **Terminate** from the menus to abort the debugging session and DS the MCP task. Then close the Debug Perspective using **Window** > **Close Perspective**.
- Back on the MCP Perspective, do the following:
  - Right-click on the program element and open its properties.
  - Select the Run-Debug Settings node in the left-hand tree.
  - Select Launch Configuration in the right-hand pane and open it (click the **Edit…** button).
  - Replace the TADS port number in the Launch Configuration with the one of your choice.
  - Close the Launch Configuration and Properties dialogs.
- Make sure your selected port is open in your firewall configuration.
- Now start the debugging session again, as above.



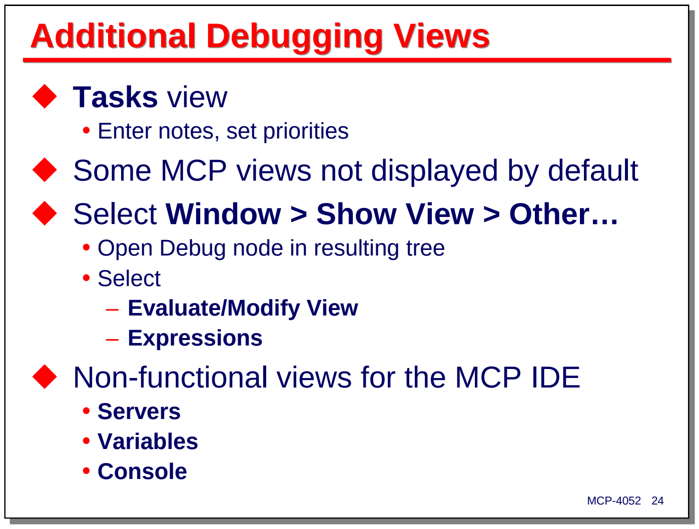
Debug Perspective Views

- **Debug** view
  - Shows task stack history (**`PERFORM`** and **`CALL`**)
- **Breakpoints** view
  - Show and enable/disable breakpoints
- **TADS Chat** view
  - Enter TADS commands and view messages
- **Editor** pane
  - Displays source file and current execution location
- **Outline** view
  - Alphabetic list of identifiers in program
  - Only effective if compiled with XREF option

MCP-4052 23

When you start a debugging session, the Debug Perspective opens. It has its own set of views, some of which are common to the MCP Perspective:

- **Debug** view – this shows the task(s) running for your debug session, along with the stack history for each one. The stack history shows you where your program is currently executing and the chain of subroutine calls and **`PERFORM`**s that are currently active.
- **Breakpoints** view – this shows a list of the breakpoints you have set for the program. We will discuss breakpoints and how they are used shortly.
- **TADS Chat** view – the IDE debugging interface is really just a front end for the TADS command line interface. It generates TADS commands and parses its responses. Those commands and responses are shown in this view as they occur. You can also enter your own TADS commands in this view and see the responses they generate.
- **Editor pane** – this is similar to the Editor tile on the MCP Perspective. You can use this to scroll through the program, view source code, set breakpoints, search for identifiers, etc.
- **Outline** view – this view presents an alphabetical list of identifiers in the program. If you click on one of the identifiers in the program, the Editor pane will scroll to the point where that identifier is declared. This view is only usable if the program has been compiled with the **`$XREFFILES`** option and the XREF files have been associated with your IDE project. There are additional Editor features available if you have XREF files, but those are beyond the scope of this presentation.



Additional Debugging Views

- **Tasks** view
  - Enter notes, set priorities
- Some MCP views not displayed by default
- Select **Window > Show View > Other…**
  - Open Debug node in resulting tree
  - Select
    - **Evaluate/Modify View**
    - **Expressions**
- Non-functional views for the MCP IDE
  - **Servers**
  - **Variables**
  - **Console**

MCP-4052 24

Continuing the discussion of Debug Perspective views…

- **Tasks** view – this is the same as the view on the MCP Perspective.

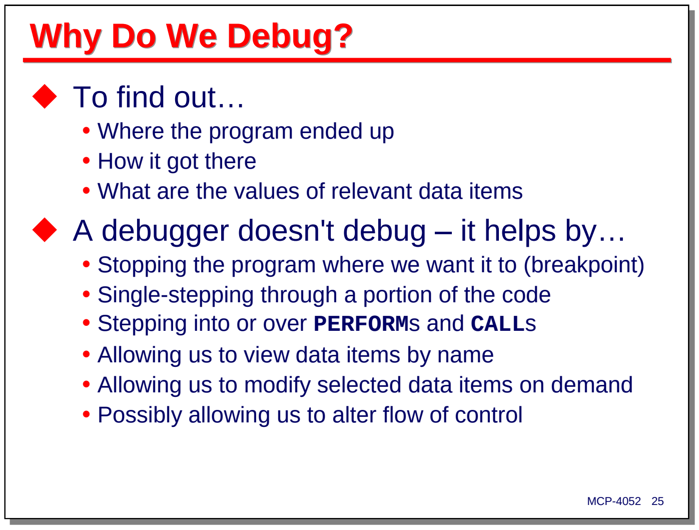
Note that some of the MCP-related debugging views are not displayed by default. To open these in the Debug Perspective:

- Select **Window** > **Show View** > **Other…**
- Open the **Debug** node in the resulting list of views
- Select **Evaluate/Modify View** and **Expressions** from the **Debug** choices.

Finally, there are three views on the standard Debug Perspective that do not appear to be functional with MCP projects. These are:

- **Servers**
- **Variables**
- **Console**

You may wish to close them when you start a debugging session so they do not take up space.



Why Do We Debug?

- To find out…
  - Where the program ended up
  - How it got there
  - What are the values of relevant data items
- A debugger doesn't debug – it helps by…
  - Stopping the program where we want it to (breakpoint)
  - Single-stepping through a portion of the code
  - Stepping into or over **`PERFORM`**s and **`CALL`**s
  - Allowing us to view data items by name
  - Allowing us to modify selected data items on demand
  - Possibly allowing us to alter flow of control

MCP-4052 25

Before we start working with a debugging session, we should pause briefly and talk about what we are actually trying to do. Generally, you need to debug a program when it does something it shouldn't have, or didn't do something it should have. There are three main questions you need to ask:

- Where did the program end up (or what path did it take through a certain area of the code)?
- How did it get there?
- What are the values of the relevant data items it was using?

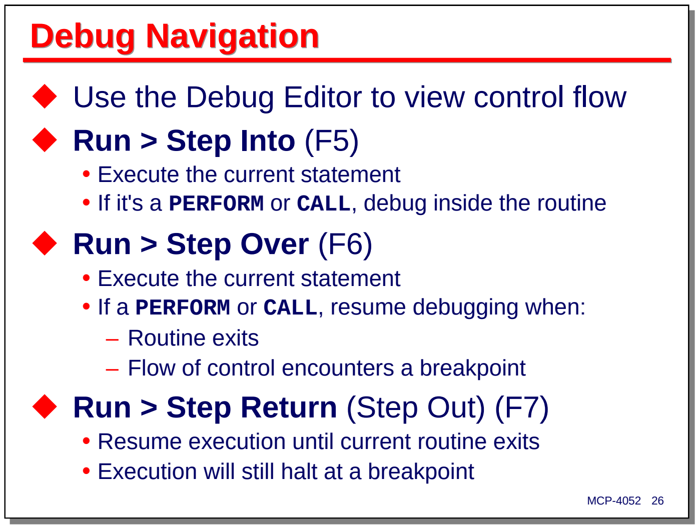
Debugging is basically the process of getting answers to those questions, and then trying to make sense of them, based on the way that the program is written.

Note that a debugger doesn't do any debugging, just as a hammer doesn't pound nails. It is just a tool you use to help you accomplish certain tasks. In the case of a debugger, it helps you to get the information necessary to answer those three main questions.

The main things a debugger allows us to do are:

- Stop the program where we want it to, either so that we can skip forward to a point of interest in the execution or examine data items at that point in the flow of control. The points at which we instruct the debugger to stop are called *breakpoints*. Some breakpoints can be just positions in the code; some may be triggered by dynamic evaluation of a condition.
- Single-step through a portion of the code. This allows you to monitor the flow of control closely, observing the path the program takes.
- Step into or over subroutines – **`PERFORM`**s or **`CALL`**s in COBOL. Stepping into a subroutine allows us to trace the flow of control inside that subroutine. Stepping over a subroutine acts like setting a breakpoint at the subroutine's exit. You typically step over when you are not interested in the details of the subroutine's operation and just want to continue stepping through the calling routine.
- View data items by the names of their identifiers. A good debugger will also allow us to compute indexing expressions and format the data in meaningful ways.
- Modify selected data items on demand. Sometimes in the process of debugging, we find where the program did something incorrectly, and we would like to override what it did and continue debugging. Modification allows us to do that. Generally you can only do that when the program has stopped at a breakpoint.
- Alter the flow of control. Sometimes this can be accomplished by modifying data on which decisions are made. Some debuggers (including COBOL-85 TADS) allow you to **`PERFORM`** routines manually.

Based on these capabilities, we should be able to monitor the way the program is working and understand how it is manipulating data.



Debug Navigation

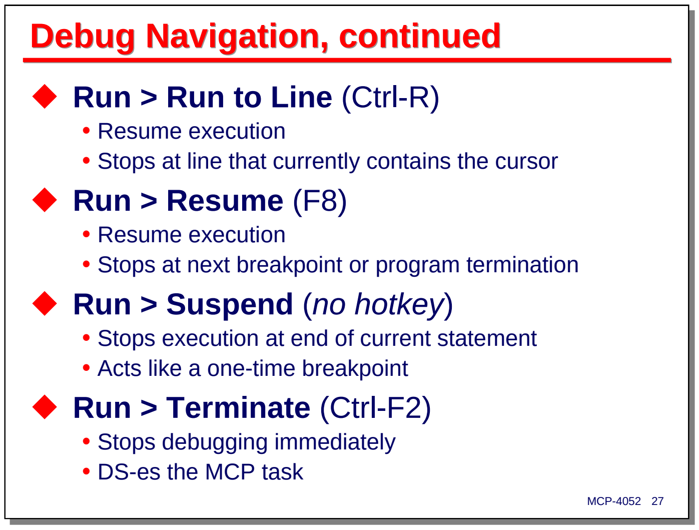
- Use the Debug Editor to view control flow
- **Run > Step Into** (F5)
  - Execute the current statement
  - If it's a **`PERFORM`** or **`CALL`**, debug inside the routine
- **Run > Step Over** (F6)
  - Execute the current statement
  - If a **`PERFORM`** or **`CALL`**, resume debugging when:
    - Routine exits
    - Flow of control encounters a breakpoint
- **Run > Step Return** (Step Out) (F7)
  - Resume execution until current routine exits
  - Execution will still halt at a breakpoint

MCP-4052 26

The first thing to understand how to do in the Debug Perspective of the MCP IDE is navigate through the program. When you start a debugging session, the IDE will cause the program to be executed, which will in turn invoke TADS. TADS will insert an implicit breakpoint before the first executable statement in the program. In COBOL, this will be the first statement in the Procedure Division. This implicit breakpoint will allow you to set other breakpoints, configure your views the way you want them, inspect initial conditions, etc.

When you are ready, you can start the program executing. One of the ways to do that is by single-stepping. There are three ways to step:

- **Run** > **Step Into** (F5) – this executes the current statement. If the statement is a subroutine call or a **`PERFORM`**, TADS will "step into" the subroutine and stop at the first executable statement therein. You can then continue stepping inside the subroutine. If the current statement is of any other type, TADS will execute it and stop at the beginning of next statement in sequence.
- **Run** > **Step Over** (F6) – this also executes the current statement. If the statement is a subroutine call or a **`PERFORM`**, TADS will "step over" the subroutine by executing the entire subroutine without stopping, unless a breakpoint is triggered in the process. Assuming no breakpoint is triggered, TADS will stop at the statement after the call. If the current statement is of any other type, TADS will execute it and stop at the beginning of next statement in sequence, the same as for Step Into. This is useful if you are not interested in the inner workings of the subroutine and do not want to step through it.
- **Run** > **Step Return** (F7) – this is also known as "Step Out". If you are stopped at some point inside a subroutine (either because you are single-stepping or because you have triggered a breakpoint), Step Return will resume execution until the subroutine exits. TADS will stop before the statement after the call that entered that subroutine. This is useful if you have been stepping inside the subroutine and realize you do not need to follow the detailed flow within the subroutine any further. It is also useful if you accidentally Step Into a subroutine when you meant to Step Over (which happens to me all the time). As with Step Over, execution will always stop when the flow of control triggers a breakpoint, so it is possible you could stop before the actual subroutine exit.



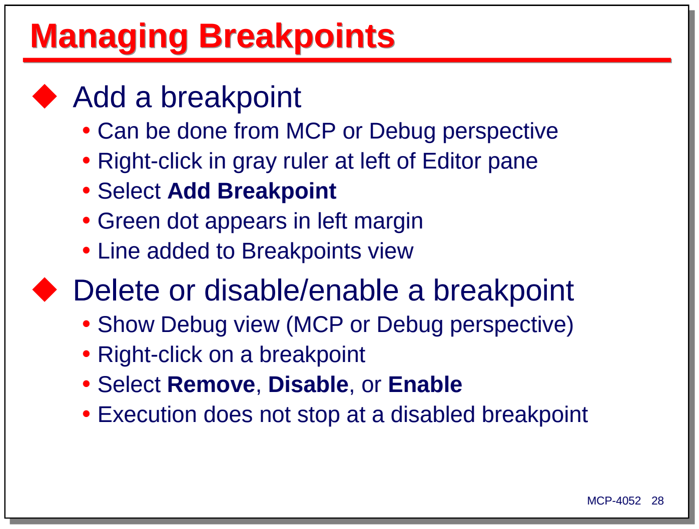
Debug Navigation, continued

- **Run > Run to Line** (Ctrl-R)
  - Resume execution
  - Stops at line that currently contains the cursor
- **Run > Resume** (F8)
  - Resume execution
  - Stops at next breakpoint or program termination
- **Run > Suspend** (*no hotkey*)
  - Stops execution at end of current statement
  - Acts like a one-time breakpoint
- **Run > Terminate** (Ctrl-F2)
  - Stops debugging immediately
  - DS-es the MCP task

MCP-4052 27

There are other ways to navigate through a program during debugging and control the execution.

- **Run** > **Run to Line** (Ctrl-R) – this is useful if you want to skip over a sequence of statements that you don't need to examine in detail and continue stepping farther along in the code. Simply scroll to the line where you want the program to stop, place the cursor in the text of that line, and select **Run to Line**. TADS will place a once-only breakpoint at the start of the line with the cursor and resume execution of the program. If there are no intervening breakpoints, execution will stop at the beginning of the statement you indicated.
- **Run** > **Resume** (F8) – this simply resumes continuous execution of the program. It will continue running until a breakpoint is triggered or the program ends.
- **Run** > **Suspend** (no hotkey) – If you have resumed the program, or for some reason it is running longer than you think it should, activating this function will stop execution, as if a breakpoint had been inserted at the point the program is currently executing. You can then examine data items, set breakpoints, single-step, resume execution, etc.
- **Run** > **Terminate** (Ctrl-F2) – this will stop debugging and terminate (DS) your program immediately. The only way to resume after this is to start a new debugging session.



Managing Breakpoints

- Add a breakpoint
  - Can be done from MCP or Debug perspective
  - Right-click in gray ruler at left of Editor pane
  - Select **Add Breakpoint**
  - Green dot appears in left margin
  - Line added to Breakpoints view
- Delete or disable/enable a breakpoint
  - Show Debug view (MCP or Debug perspective)
  - Right-click on a breakpoint
  - Select **Remove**, **Disable**, or **Enable**
  - Execution does not stop at a disabled breakpoint

MCP-4052 28

We have been discussing breakpoints and how triggering one will stop execution of a program that is being debugged, but how are they created?

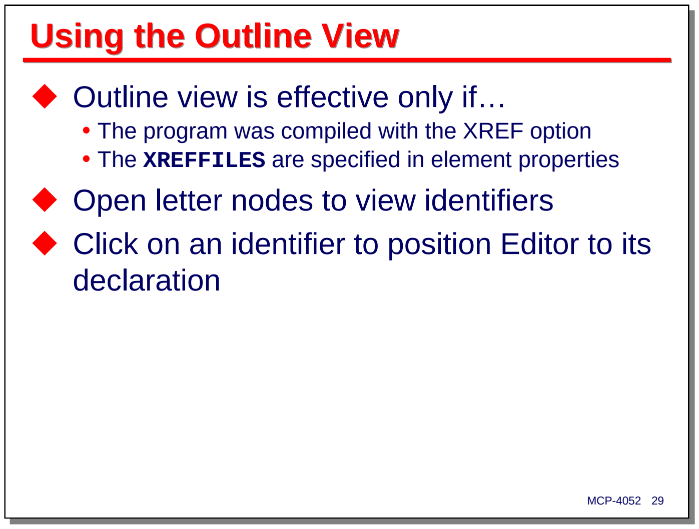
You add breakpoints on the Editor pane of either the Debug or MCP Perspective. Being able to add them on the MCP Perspective is nice, as it allows you to set up breakpoints before you start a debugging session. Breakpoints created on either perspective will be saved as part of your project's metadata, and will still be available after you have closed and reopened the project.

Begin by scrolling to the line where you want the breakpoint.

- Right-click in the gray ruler area at the left of the Editor pane
- Select **Add Breakpoint** from the context menu.
- A green dot appears in the left margin to indicate the line has a breakpoint.
- A entry is also made in the Breakpoints view.

The Breakpoints view can be shown on both the MCP and Debug perspectives. It can be used to view and manage breakpoints once they are created:

- To delete a breakpoint, right-click on it and select **Remove** from the context menu.
- To disable a breakpoint, uncheck the box next to it in the Breakpoints view. A disabled breakpoint remains associated with the program, but will not cause the program to step when it is triggered. To re-enable the breakpoint, simply check its box in the Breakpoints view.
- To navigate to the line where a breakpoint is set, double-click the breakpoint in the Breakpoints view.



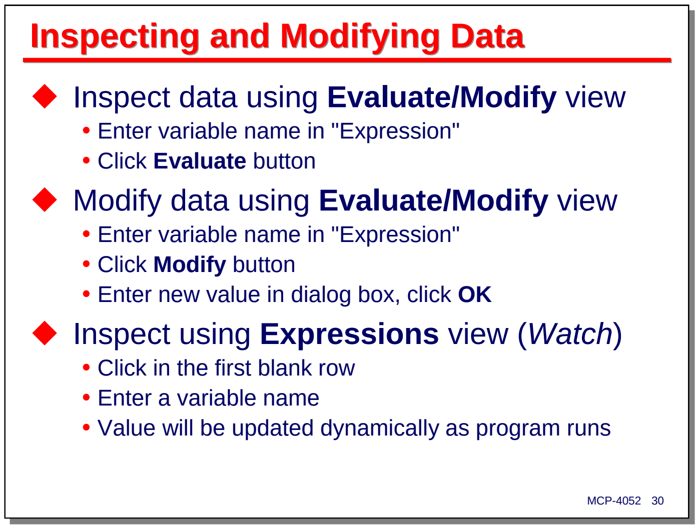
Using the Outline View

- Outline view is effective only if…
  - The program was compiled with the XREF option
  - The **`XREFFILES`** are specified in element properties
- Open letter nodes to view identifiers
- Click on an identifier to position Editor to its declaration

MCP-4052 29

The Outline view gives you a way to quickly navigate to the declaration of an identifier. It can only be used if you have compiled the program with the **`$XREFFILES`** option and the resulting XREF files have been associated with the element in your project. If you set the **Compile with Xref** option in the element properties and build from Eclipse, that association will happen automatically.

To use the Outline view, simply open the appropriate node in the alphabetical list and click on the desired identifier. The Editor pane will scroll to that identifier's declaration.



Inspecting and Modifying Data

- Inspect data using **Evaluate/Modify** view
  - Enter variable name in "Expression"
  - Click **Evaluate** button
- Modify data using **Evaluate/Modify** view
  - Enter variable name in "Expression"
  - Click **Modify** button
  - Enter new value in dialog box, click **OK**
- Inspect using **Expressions** view (*Watch*)
  - Click in the first blank row
  - Enter a variable name
  - Value will be updated dynamically as program runs

MCP-4052 30

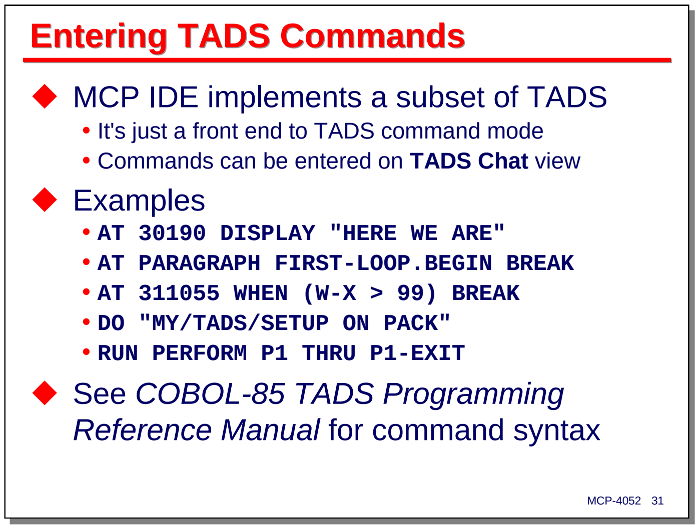
There are two views that will allow you to examine the values of data items within your program. One of them also allows you to modify values of data items.

The Evaluate/Modify view allows you to view or modify the value of a data item. It can only be used when the program is stopped, either at a breakpoint, or because you are single-stepping.

- To view the value of an item, enter its name in the **Expression** text box, and click the **Evaluate** button. The value will be shown in the **Result** text box.
- To modify the value of an item, enter its name in the **Expression** text box, and click the **Modify** button. A small dialog box will pop up into which you can enter the new value.

The Expressions view works a little differently. It can only be used to view data items, but you can set this view to monitor values dynamically. Another name for this is a *watch*. Each time that execution stops (at a breakpoint or a step), Eclipse will request from TADS all of the values currently on the Expressions list (all that are in scope, that is) and display them in this view.

You can edit and remove expressions from this view by right-clicking on them within the view.



Entering TADS Commands

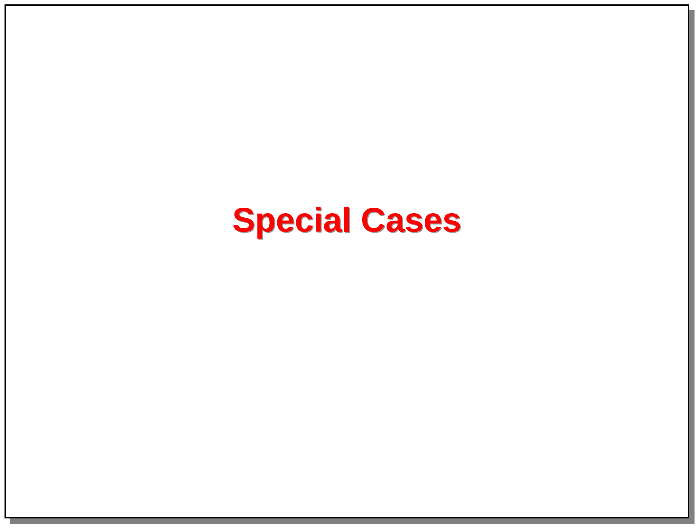
- MCP IDE implements a subset of TADS
  - It's just a front end to TADS command mode
  - Commands can be entered on **TADS Chat** view
- Examples
  - **`AT 30190 DISPLAY "HERE WE ARE"`**
  - **`AT PARAGRAPH FIRST-LOOP.BEGIN BREAK`**
  - **`AT 311055 WHEN (W-X > 99) BREAK`**
  - **`DO "MY/TADS/SETUP ON PACK"`**
  - **`RUN PERFORM P1 THRU P1-EXIT`**
- See *COBOL-85 TADS Programming Reference Manual* for command syntax

MCP-4052 31

As I mentioned earlier, the Debug view of the MCP IDE is just a front end to the command-line interface to TADS. All of the things you can do in a debugging session we have discussed thus far simply cause the IDE to generate the appropriate TADS command and submit it.

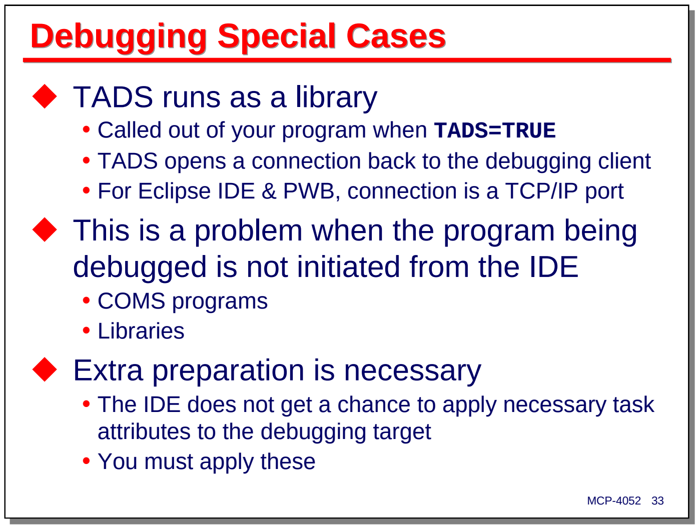
You can bypass the IDE's interface and send commands directly to TADS using the TADS Chat view. Simply type the command into the **Enter TADS** command textbox, press **Enter**, and view the response in the **Output** pane of the view.

COBOL-85 TADS has a large number of commands, and only a portion of those are directly supported by the IDE. You can read about the rest of them in the *COBOL-85 TADS Programming Reference Manual*.



Special Cases

The use of the MCP IDE for Eclipse we have discussed thus far works well for standard programs that you can directly run an control. Not all programs are like that, however, so this final section of the presentation discusses a couple of common special cases.



Debugging Special Cases

- TADS runs as a library
  - Called out of your program when **TADS=TRUE**
  - TADS opens a connection back to the debugging client
  - For Eclipse IDE & PWB, connection is a TCP/IP port
- This is a problem when the program being debugged is not initiated from the IDE
  - COMS programs
  - Libraries
- Extra preparation is necessary
  - The IDE does not get a chance to apply necessary task attributes to the debugging target
  - You must apply these

MCP-4052 33

To understand why some programs cannot be directly debugged in the MCP IDE (or in standard TADS, for that matter), consider how TADS works.

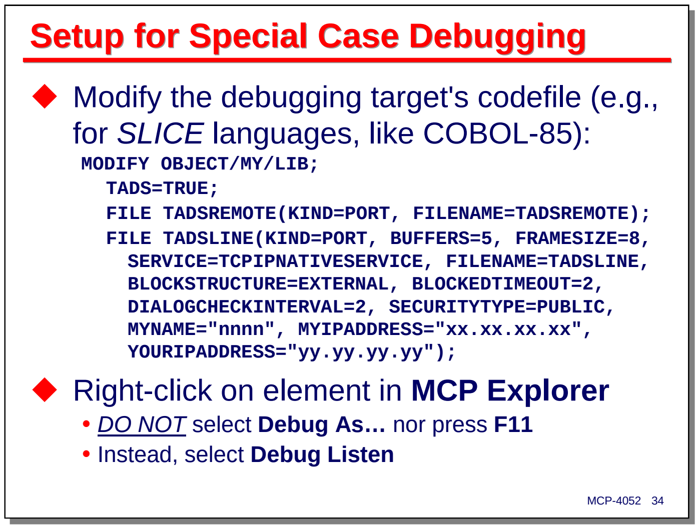
- TADS runs as a library.
- When you run your TADS-enabled program with the task attribute **TADS=TRUE**, the MCP invokes the TADS library as part of the program's initiation.
- TADS then opens a connection back to the debugging client. Standard TADS uses a remote file for its command-line interface, so in most cases, the association between that remote file and the originating datacom station happens automatically.
- For the Eclipse IDE (and PWB), the connection back to the debugging client is a TCP/IP port. Note that TADS is acting as the client (initiator) for this connection, and the debugging client (the Eclipse IDE or PWB) is acting as the server (listener) for the connection. This is why there are potentially firewall issues when using either Eclipse or PWB.
- The Eclipse IDE and PWB apply task attributes to the program that they directly initiate so that TADS can connect to the appropriate host and port. In particular, the IDEs specify **TADS=TRUE**, and file-equate the IP address and port numbers the TCP/IP port files are to use.

This arrangement is not a problem when the program is one that the Eclipse IDE can run directly, but there are two common cases where it will not work without manual intervention:

- COMS programs
- Library programs

The reason is that these types of programs are not initiated directly. COMS programs are, of course, initiated by COMS; Library programs are initiated by the MCP when a calling program makes first reference to them. The problem is that the Eclipse IDE does not get a chance to supply the TADS task attribute or file-equation information that allows TADS to connect to the debugging client.

To get around this problem, you must supply the task attributes to the COMS or Library program that is the debugging target. You must also initiate the debugging session a slightly different way. The next two slides explain how to do this.



Setup for Special Case Debugging

- Modify the debugging target's codefile (e.g., for *SLICE* languages, like COBOL-85):

```
MODIFY OBJECT/MY/LIB;
  TADS=TRUE;
  FILE TADSREMOTE(KIND=PORT, FILENAME=TADSREMOTE);
  FILE TADSLINE(KIND=PORT, BUFFERS=5, FRAMESIZE=8,
    SERVICE=TCPIPNATIVESERVICE, FILENAME=TADSLINE,
    BLOCKSTRUCTURE=EXTERNAL, BLOCKEDTIMEOUT=2,
    DIALOGCHECKINTERVAL=2, SECURITYTYPE=PUBLIC,
    MYNAME="nnnn", MYIPADDRESS="xx.xx.xx.xx",
    YOURIPADDRESS="yy.yy.yy.yy");
```

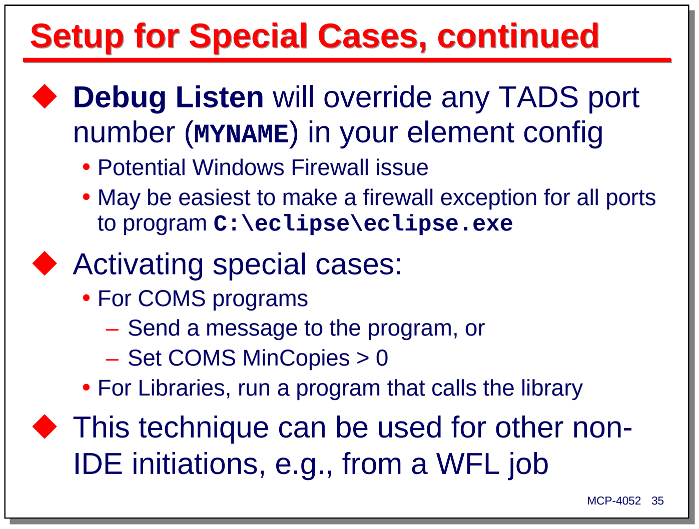
- Right-click on element in **MCP Explorer**
  - *DO NOT* select **Debug As...** nor press **F11**
  - Instead, select **Debug Listen**

MCP-4052 34

The easiest way to prepare a COMS or Library program for debugging is to do a WFL MODIFY on its codefile. This means you will need to compile the program first with the **$TADS** option set.

The task attributes that need to be set vary slightly depending on whether the program is compiled with one of the so-called SLICE compilers (COBOL-85, CC, FORTRAN-77, PASCAL-83) or a non-slice compiler. The example shown on the slide is for the SLICE compilers. The attributes for non-SLICE languages can be found in the PWB Help file.

The second part of the setup for special cases is to initiate the debugging session in a different way. When you are ready to debug, do not select **Debug As…** from the element's context menu or click F11. Instead, select **Debug Listen** from the element's context menu. This will cause the Eclipse IDE to simply wait for the TADS connection without attempting to run the program first.



Setup for Special Cases, continued

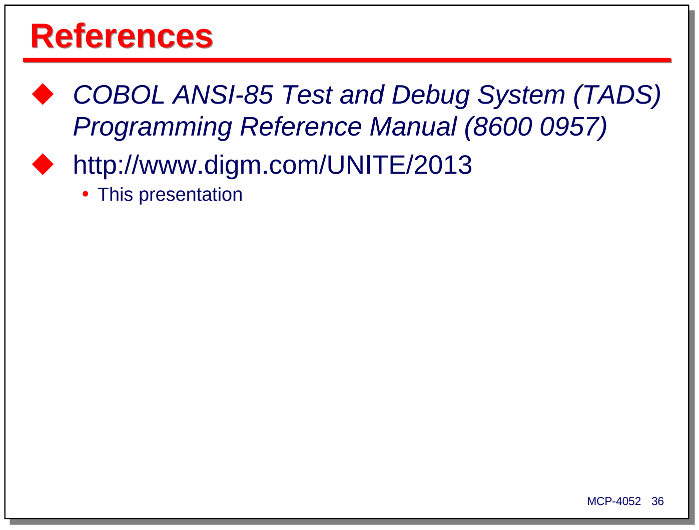
- **Debug Listen** will override any TADS port number (**`MYNAME`**) in your element config
  - Potential Windows Firewall issue
  - May be easiest to make a firewall exception for all ports to program **`C:\eclipse\eclipse.exe`**
- Activating special cases:
  - For COMS programs
    - Send a message to the program, or
    - Set COMS MinCopies > 0
  - For Libraries, run a program that calls the library
- This technique can be used for other non-IDE initiations, e.g., from a WFL job

MCP-4052 35

Note that activating Debug Listen will override any TADS port number (i.e., the port file **`MYNAME`** attribute) for your element. This is another potential firewall issue, and generally you won't be able to get by with opening just one port for TADS debugging. In this case, it is probably easier to simply enable the **`eclipse.exe`** program to accept connections on any port.

The final step in the setup for special-case debugging is to initiate the program you want to debug externally. For a COMS program, that means you need to send a message to the program, or set its **`MinCopies`** attribute to a non-zero value in the COMS configuration. For a library, you need to run some program that calls the library. The **Run** tab of the element's **MCP Properties** allows you to specify a client program that can be run to call your library.

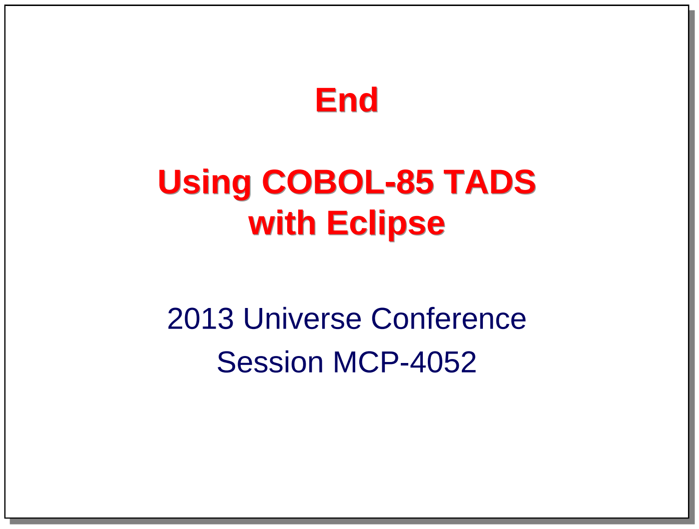
Note that this technique can be used with any type of program that you don't want to run directly out of the IDE. For example you could debug a program that is initiated by a WFL job using this same approach.



References

- *COBOL ANSI-85 Test and Debug System (TADS) Programming Reference Manual (8600 0957)*
- http://www.digm.com/UNITE/2013
  - This presentation

MCP-4052 36



End

Using COBOL-85 TADS with Eclipse

2013 Universe Conference

Session MCP-4052