Using CCF

2005 UNITE Conference
Minneapolis, Minnesota

Session MCP-4017

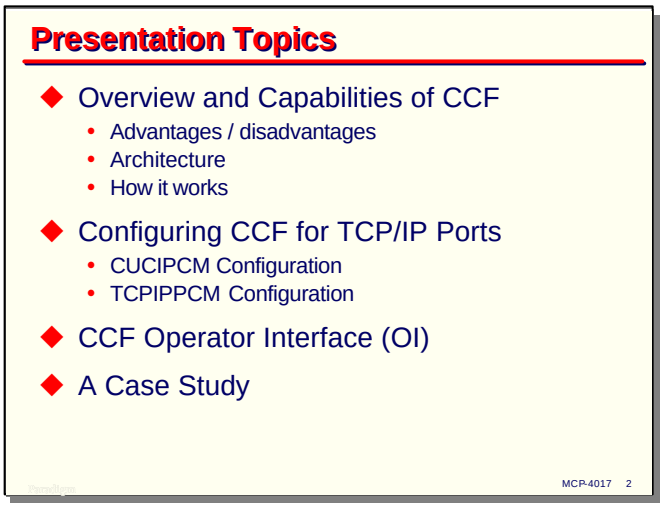
Tuesday, 18 October 2005, 2:45 p.m.

Paul Kimpel

Paradigm Corporation
Poway, California

http://www.digm.com

e-mail: paul.kimpel@digm.com

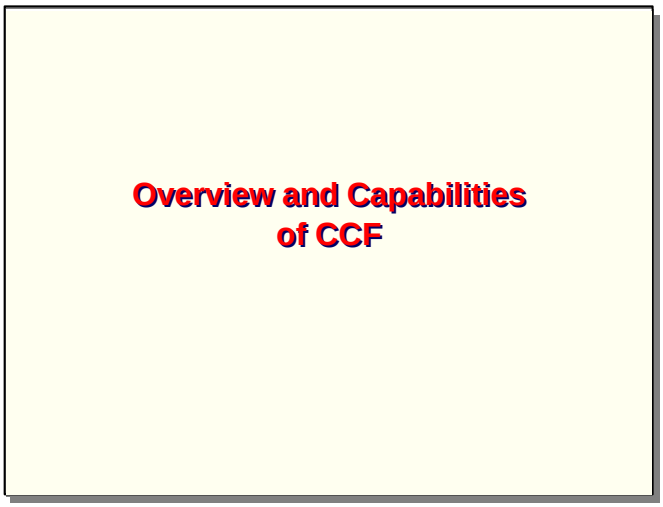


Today I want to talk about one of my favorite pieces of software, CCF. This is one of the most useful, but perhaps least appreciated, elements of the ClearPath MCP software suite. I will begin with an overview of CCF and discuss its capabilities, along with a brief discussion of its advantages and disadvantages, its architecture, and a high-level picture of how it works.

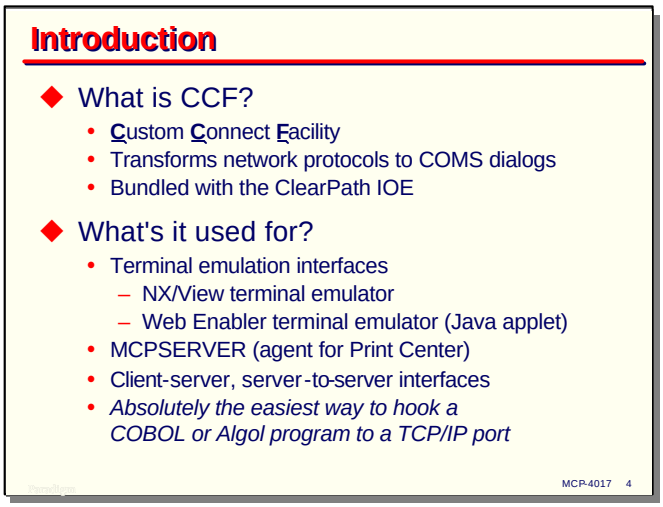
CCF has a number of facets, but the one I will focus on in this presentation is configuring it for use with TCP/IP ports. That process has two primary parts, one for a module called the CUCIPCM and one for a module called the TCPIPPCM.

Next, I will review of the CCF operator interface (OI) and discuss the commands that can be used with it at run time.

Finally, we will look at a case study showing how CCF can be applied to a real problem.



Let us begin by way of an overview of CCF and its capabilities

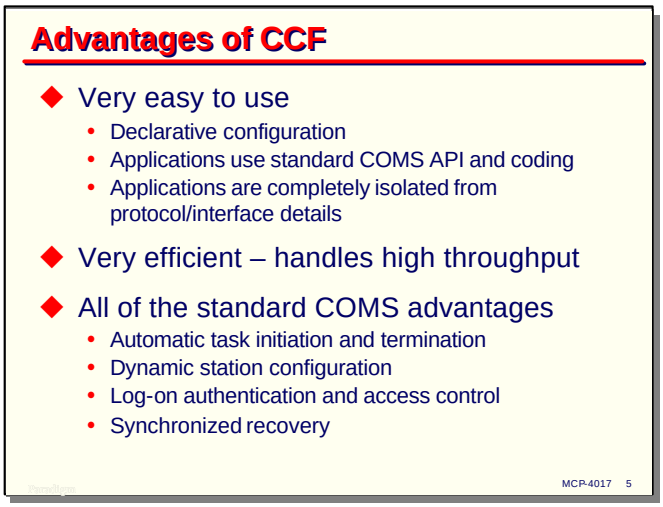


What is CCF? The acronym stands for **C**ustom **C**onnect **F**acility. The primary thing CCF does is transform connections to an MCP system established through network protocols into COMS station dialogs. In other words, it allows these network connections to appear to COMS as if they were datacom stations. The standard implementation includes support for TCP/IP ports, named pipes, and HTTP, the web browser protocol.

CCF is bundled with the ClearPath Integrated Operating Environment (IOE), so if you have a ClearPath MCP system, you have CCF. In addition, you have probably been using it, whether you are aware of that or not.

So what is CCF used for? The answer is a number of things.

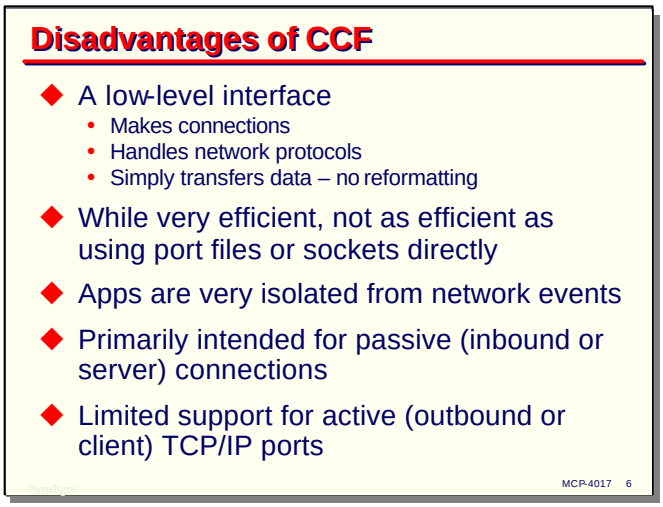
- The original, and still one of the most common uses for CCF is terminal emulation. CCF is the mechanism used by NX/View and Web Enabler (the Java applet) to connect to MCP systems and establish dialogs with COMS
- CCF is the mechanism by which the Windows-based Print Center utility communicates with its MCP-resident agent program, MCPSERVER.
- CCF is generally a good choice for implementing client-server and server-to-server interfaces.
- Overall, CCF is absolutely the easiest way to hook an MCP COBOL or Algol program to a TCP/IP port and have that program communicate with other systems over a network.



CCF has a number of advantages. First among these is that it is very easy to use. CCF has a simple declarative configuration. Typically there is no programming involved beyond what you would do in a normal COMS program. Your MCP-resident applications use the same COMS API and coding techniques they always have. Network connections through CCF appear as normal COMS stations both to COMS and to your COMS TPs. The COMS TPs are isolated completely from the network, and normally need not be concerned with details of establishing connections, error handling, or other network issues.

CCF is also very efficient. It has low overhead and can handle high volumes of traffic.

Because applications using CCF are normal COMS programs, you retain all of the standard COMS advantages, including automatic task initiation and termination, dynamic station configuration, log-on authentication against the USERDATAFILE, COMS access control facilities, and DMSII synchronized recovery.

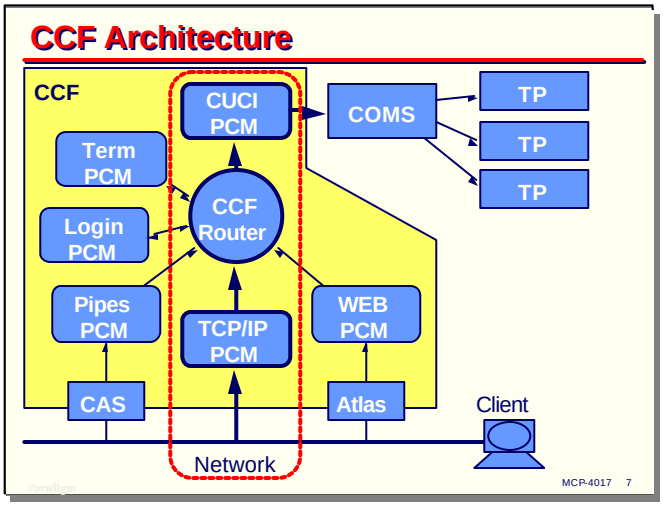


CCF has a few disadvantages as well. First, it is a relatively low-level interface. It handles the establishment of network connections, handles the network protocols, and transfers data in and out of COMS. That's it – it has no other formatting or processing capabilities itself, although such functions could be implemented in CCF PCMs, as we will see shortly.

While CCF is very efficient, it is not as efficient as using port files, sockets, or named pipes directly. On the other hand, programming for these lower-level interfaces can be quite challenging, and CCF usually makes up in convenience and reliability the small amount it gives up in overhead.

I mentioned on the prior slide the advantage that CCF isolates the MCP-resident programs from the details of the network. That can also be a disadvantage in some applications, especially if your application needs to be sensitive to network events, such as timeouts or blocked transmissions.

CCF is intended primarily for supporting passive connections. That is, connections which originate outside the MCP, where some other system is acting as the client and the MCP system is acting as the server. With one exception, you cannot use CCF to originate connections from the MCP to an external system. That exception is TCP/IP ports, for which there is some limited support for establishing outbound connections.



CCF has a very interesting, open-ended architecture. This diagram illustrates the relationship of the modules which come with the standard release.

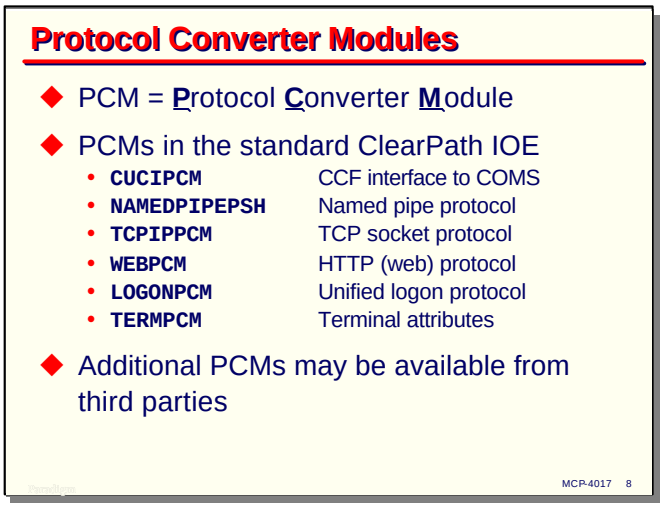
At the core of CCF is a module called the Router. Its job is to connect the other CCF modules and route the processing of connections among them, hence its name.

Then there are a series of network protocol interface modules. The standard release has three of these, The TCPIPPCM, which handles TCP/IP connections, the NAMEDPIPEPSH, which works with Client Access Services to handle named pipe connections, and the WEBPCM, which works with Atlas (the Web Transaction Server product) to handle HTTP connections from web browsers.

At the other end of the Router is a module named the CUCIPCM. Connections from the network protocol modules are eventually routed to CUCIPCM, which is the interface into COMS.

There are two optional utility modules in the standard release, The LOGONPCM and TERMPCM, which I will discuss in more detail shortly.

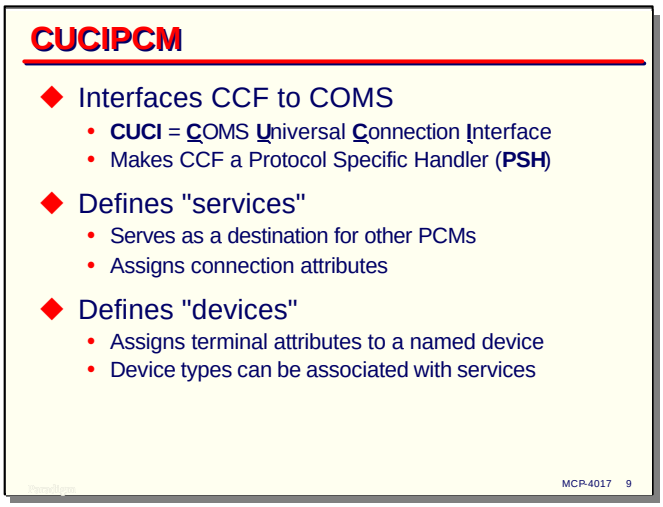
For this presentation, I am going to focus on configuring TCP/IP connections using the TCPIPPCM and CUCIPCM, as this type of connection is frequently used for client-server and server-to-server applications.



What are all those PCMs and what do they do? PCM stands for Protocol Converter Module. As we saw, there are six of them in the standard ClearPath software release. Note that the NAMEDPIPEPSH is a PCM, even though its name does not indicate that.

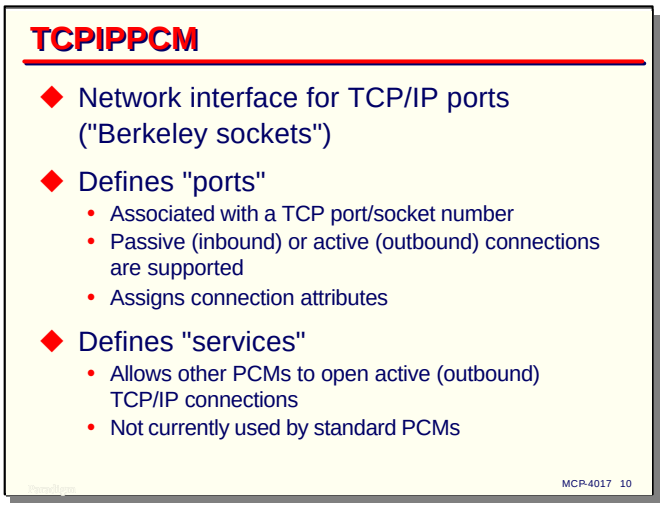
CCF has an open-ended design, so it is possible to have additional PCMs that would handle network protocols or provide utility functions. CCF was originally developed by the Stahura-Brenner Group, and in the past they have offered their PCMs for license.

In the following slides I will describe each of the standard PCMs and briefly discuss what they do.



As mentioned previously, the CUCIPCM is the interface between CCF and COMS. "CUCI" stands for COMS Universal Connection Interface, and is the name of a Unisys-developed standard for interfacing external protocols to COMS. A software element that implements this CUCI standard is termed a Protocol Specific Handler, or PSH. There are a number of standard PSHes in addition to CCF, including Telnet, the TCP Print Enabler, and the ConnectorPSH used with MQ and COMTI.

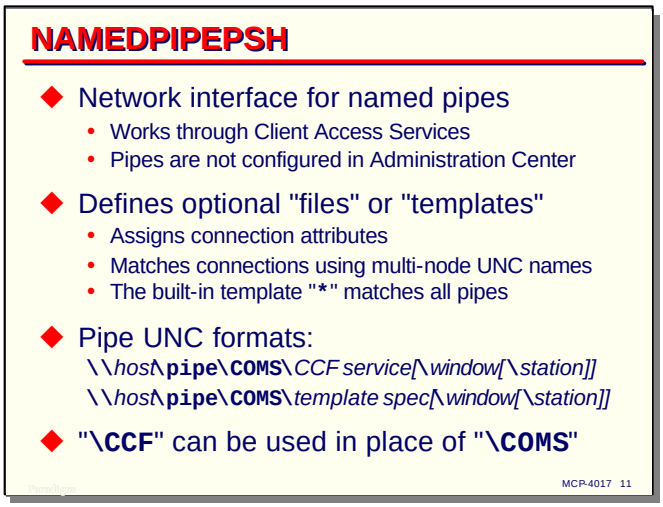
In the context of the CUCIPCM, you define two types of entities, "services" and "devices." A service acts as a destination to which other PCMs can route connections. A service also provides a means to associate values of attributes with a connection. A device is very much like a terminal definition in the old NDL and NDLII configurations for classic datacom networks. It provides a way to specify values for attributes of physical devices and assign a name to groupings of such attributes. Once defined, device names can be associated with CCF services.



The TCPIPPCM is the network interface for the TCP/IP protocol. Connections through this protocol are termed ports, or sometimes Berkeley Sockets.

In the context of the TCPIPPCM, you define two types of entities, "ports" and "services." Ports are associated with a specific TCP port or socket number, usually in the range of 1024-65535. TCPIPPCM ports always specify another CCF service as a destination, and can assign attribute values to a connection.

TCPIPPCM services provide a mechanism for other PCMs to open active, or outbound, TCP/IP connections. The documentation on this feature is not very revealing, I know of no way this capability can be used by the existing standard PCMs, and thus will not discuss it further in this presentation.



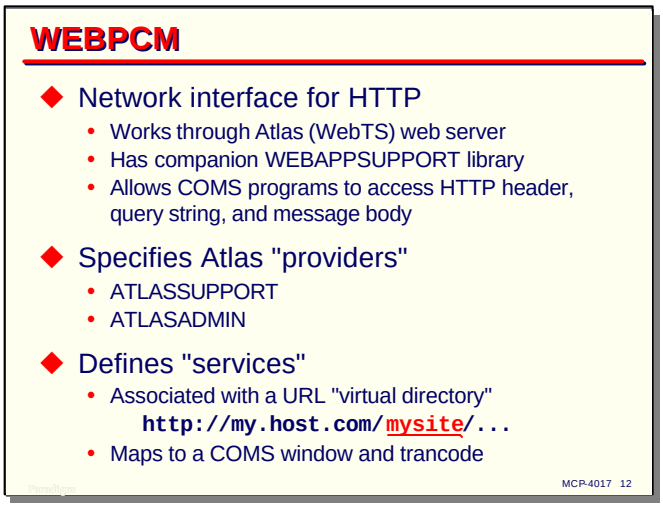
The NAMEDPIPEPSH is another network interface PCM. It works through the MCP's Client Access Services facility to interface the Windows named pipe protocol to CCF.

In the context of the NAMEDPIPEPSH, you define entities called "files" or "templates" (the two terms are synonymous). Named pipes are established by the client using a Universal Naming Convention (UNC) string. The file or template specification in CCF defines a suffix to the UNC (or a wildcard pattern for the suffix).

When the client connects, they open a pipe with a multi-node string containing "****" characters separating the nodes. The string begins with "****" followed by the domain name or IP address of the MCP host. The next node must be "**\pipe**", followed by either "**\COMS**" or "**\CCF**". Following that node there can be one of two types of nodes:

- The name of a CCF service. This will be the next service within CCF to which the connection will be routed. This name optionally can be followed by a COMS window name and a station name to be assigned to the connection.
- A template specification. This can also optionally be followed by a window and station name.

It is not necessary to define templates within CCF at all. There is a default template, "*****", which matches all UNC template specifications not otherwise defined in CCF. This is what allows the fourth node in the name to be a CCF service name. This default template does not appear in the standard CCF configuration, but it could be included in order to assign default attributes to it.

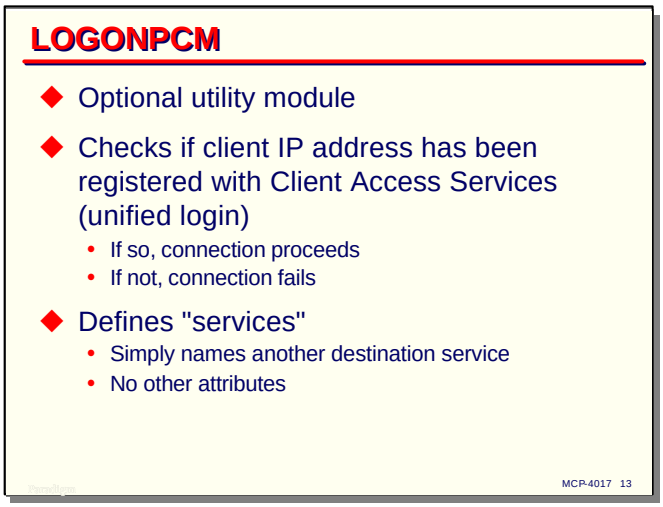


The WEBPCM is a network interface module for the HTTP protocol, which normally operates over TCP/IP port 80. Like the NAMEDPIPEPSH, the WEBPCM does not handle these network connections directly, but works through the MCP Web Transaction Server, Atlas. The WEBPCM is implemented as an Atlas AAPI library.

The WEBPCM typically requires additional programming in the COMS TPs to handle the peculiarities of HTTP interfaces and HTML formatting. To support this, the WEBPCM has a companion support library, WEBAPPSUPPORT, which allows COMS programs to access fields in the HTTP header, query string, and message body. This library also has facilities to aid the generation of HTML output.

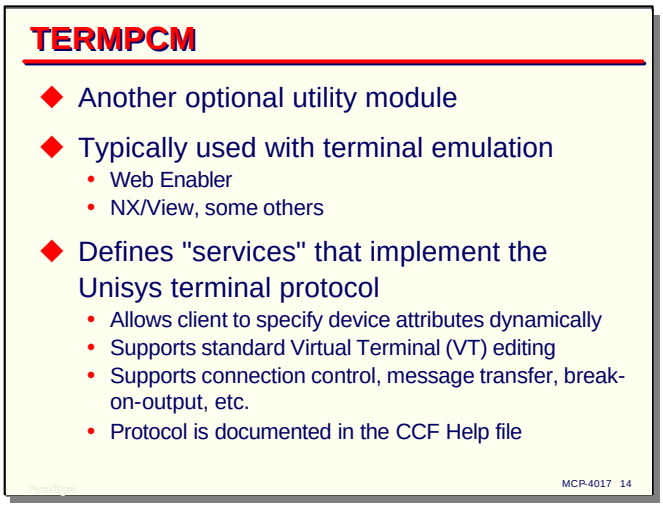
In the context of the WEBPCM, you define "providers" and "services." A provider corresponds to an instance of the Atlas web server. ClearPath systems come with two such instances by default, **`ATLASSUPPORT`** and **`ATLASADMIN`**. A site can add to these if needed.

A WEBPCM service is associated with a virtual directory node in incoming URLs. The virtual directory node is the one following the host name or IP address in an HTTP URL. The WEBPCM service maps the URL to a COMS window and trancode, and may supply additional connection attributes. The destination for a WEBPCM connection is typically the CUCIWEBSERVICE in the CUCIPCM, but other destinations can be specified.



The LOGONPCM is an optional utility module. It simply checks if the client IP address for a connection has been registered with Client Access Services. If it has, the connection is allowed to proceed; if not, the connection fails. This PCM implements unified logon for CCF connections. It is normally used with named pipes, and is what allows NX/View to automatically log a client PC on to the MCP using the client's Windows user name and password.

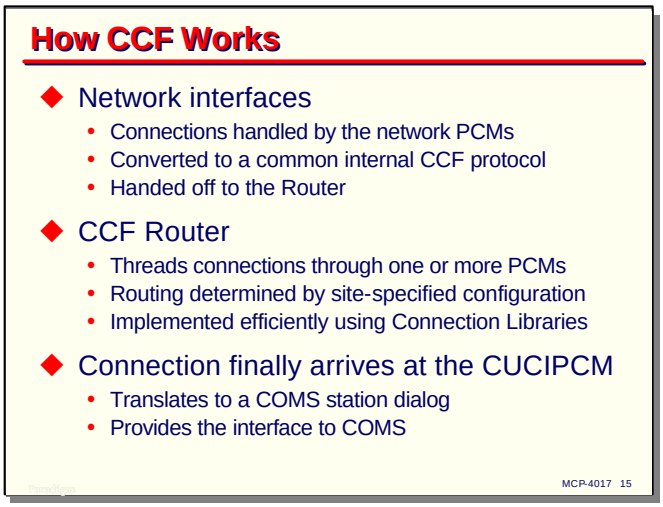
The only thing you can configure for the LOGONPCM is a service, which simply names another service as a destination. No other connection attributes can be specified by this PCM.



The TERMPCM is another optional utility module. The purpose of this PCM is to implement the Unisys terminal protocol. This module is typically used with terminal emulation clients. In particular, it is used with the Web Enabler Java applet and NX/View.

Within the context of the TERMPCM, you define "services" that implement the terminal protocol. These services associate default attribute values with a connection, and specify another service as a destination for the connection's routing.

The terminal protocol runs on top of whatever network protocol is being used. It allows the client to specify device attributes dynamically when a connection is established. It also provides support for virtual terminal editing, connection control, message transfer, break-on-output, and other traditional datacom functions. The protocol is documented in the CCF Help file.

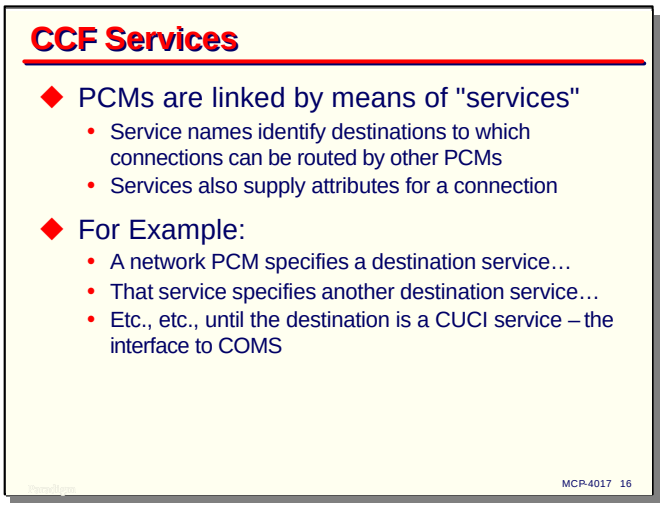


With that overview of the standard modules available with CCF, let us examine how CCF works.

Connections from external systems are handled by the network interface PCMs. These PCMs deal with the specifics of the individual network protocols and then convert the connection to a common internal CCF protocol. The converted connection is then handed off to the CCF Router.

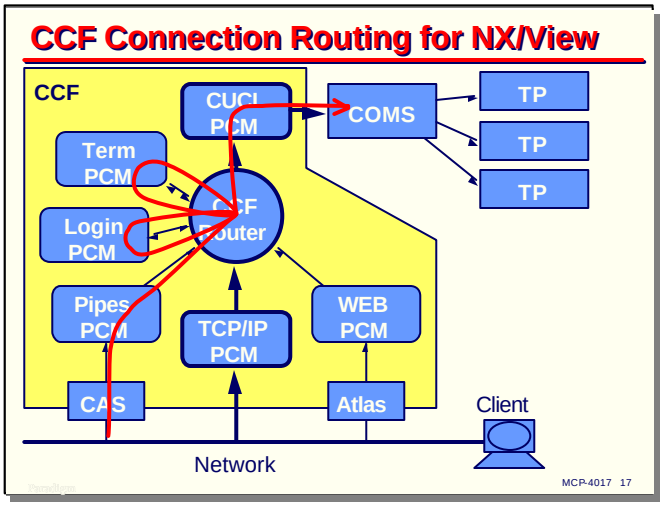
Based on the site-specified configuration, the Router threads the connections through one or more PCMs. This routing is implemented very efficiently through the use of Connection Libraries.

The routing for the connection eventually arrives at the CUCIPCM. Here the connection is translated to a COMS station dialog and the CUCIPCM provides the interface to COMS.



As we have seen, many of the PCMs define "services." In CCF, services are the means by which connections are routed through the system. Each service has an identifying name. A PCM entity names a service as its destination, and the Router uses this information to establish the pathway for a connection through CCF. Services are also a mechanism to supply values for attributes of the connection as it passes through its series of PCMs.

For example, a network PCM handles an incoming connection. It specifies a service as its destination. That service specifies another destination service. This continues until the destination is a CUCIPCM service. That is the interface to COMS, so that terminates the routing.



This diagram illustrates the routing for an NX/View named-pipe connection.

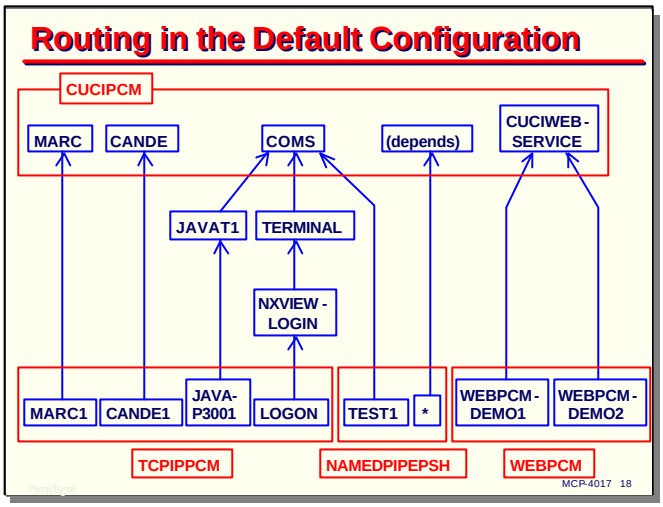
The connection is handed to the NAMEDPIPEPSH PCM by Client Access Services. The named pipe PCM translates the connection to the internal CCF protocol and hands it off to the Router. The router consults the CCF configuration tables and routes the connection to the LOGONPCM for user authentication.

If the IP address is already registered by Client Access Services, the LOGONPCM signals success, and the Router next sends the connection to the TERMPCM.

The TERMPCM strips off the terminal protocol envelope and acts on it.

Any resulting message text is then sent by the Router to the CUCIPCM, where it becomes an input message to COMS.

On output, messages from COMS follow the same path in reverse, until the data is handed off to Client Access Services for transmission back to the client.



This diagram shows how connections are routed in the default configuration that comes with the system release. At the top are the CUCIPCM services, which are the final destinations for connections before they are sent to COMS.

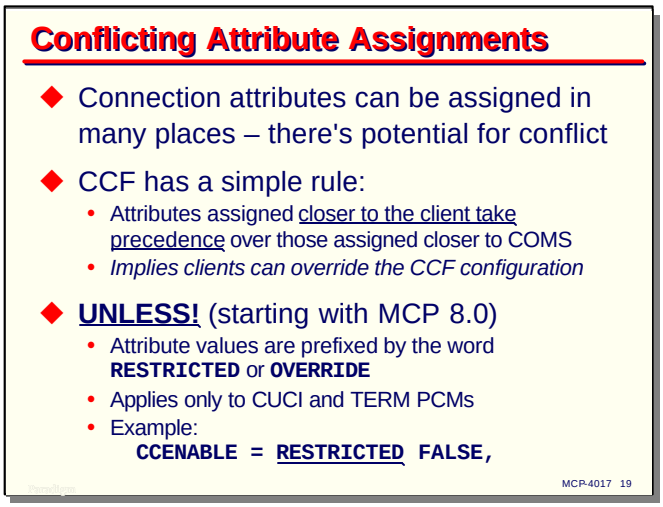
Along the bottom are a series of network interfaces. Within the TCPIPPCM, **MARC1** and **CANDE1** are ports that operate on TCP port numbers 2001 and 2002, respectively. They are simply routed to CUCIPCM services **MARC** and **CANDE**, which are associated with the COMS windows of the same name.

The **JAVAP3001** port is used by Web Enabler. It is routed to TERMPCM service **JAVAT1**, and from there into the CUCIPCM service **COMS**.

The **LOGON** port is used by NX/View when operating over TCP/IP ports rather than named pipes. Its connections are routed through the LOGONPCM service **NXVIEWLOGIN**, followed by the TERMPCM service **TERMINAL**, before arriving at the CUCIPCM service **COMS**.

The NAMEDPIPEPSH template **TEST1** is simply routed to the CUCIPCM service **COMS**. The default named pipe template, "*****", does not specify a destination service in the standard configuration file, so the destination for its connections will depend on the fourth node of the UNC name used to establish the connection.

Finally, there are two demonstration services for the WEBPCM. These are routed to the CUCIPCM service **CUCIWEBSERVICE**.

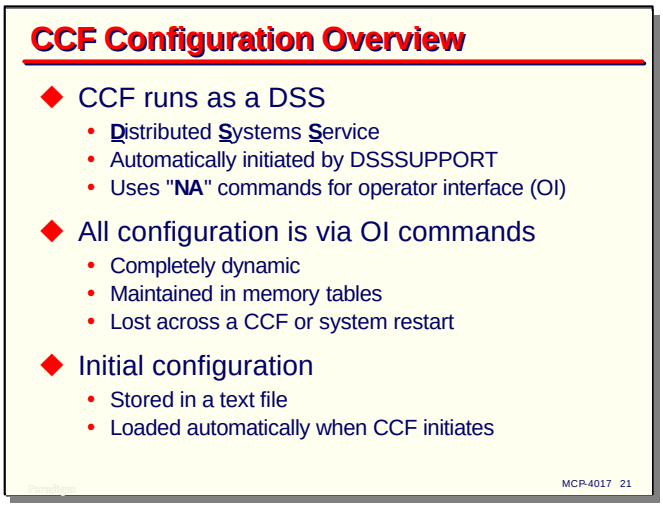


As you may have noticed from the discussion thus far, attribute values can be assigned to connections by almost any of the PCMs. Since attributes can be assigned in many places along a connection's path through CCF, there is a potential for conflicting assignments in different PCMs.

CCF has a simple rule to resolve attribute conflict. Assignments made along the connection's path that are closer to the client take precedence over those that are assigned closer to COMS. That is, assignments made earlier cannot be overridden by those made later along the path. This implies that client assignments can override those in the CCF configuration on the MCP host.

It was eventually recognized that giving clients absolute authority to override things like their station name and COMS window is not always a good thing to do, so starting on the MCP 8.0 release, this precedence was modified somewhat. In the CUCI and TERM PCMs, you can include the word **RESTRICTED** or **OVERRIDE** in front of an attribute value. That causes that attribute to be overridden by the value in this PCM, regardless of its setting earlier along the connection path. This override only applies to the CUCIPCM and TERMPCM, not to any of the network interface PCMs.

With that overview of CCF, now let us turn to how CCF is configured.

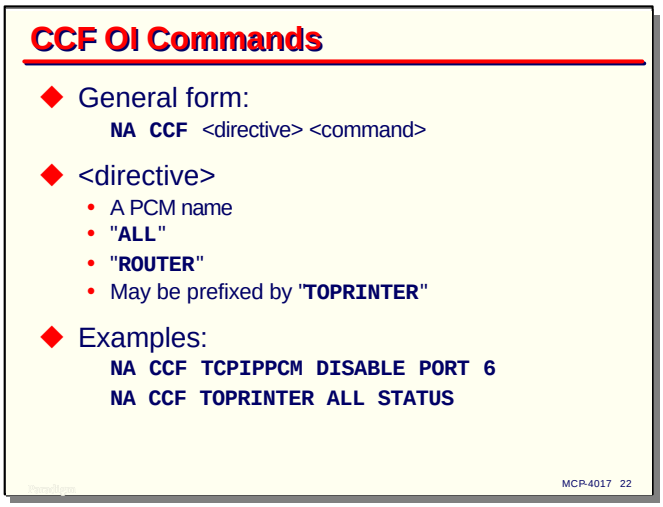


CCF runs in the MCP environment as a DSS – a Distributed Systems Service. As such it is initiated automatically by DSSSUPPORT (although a site can change this) and the operator interface (OI) is through "NA" ODT commands.

All configuration and administration of CCF is done through the operator interface. The configuration is completely dynamic and can be changed while CCF is running. The configuration data is maintained in memory tables. This means that the current configuration will be lost across a CCF or system restart.

An initial configuration, however, can be stored in a text file. This text file is loaded automatically when CCF initiates. For most sites, the text file stores the entire configuration – dynamic changes to the configuration are usually rare.

This initial configuration can be modified at run time, but there is presently no mechanism to dump the running configuration to a text file. You must keep track of changes and update the text file yourself.



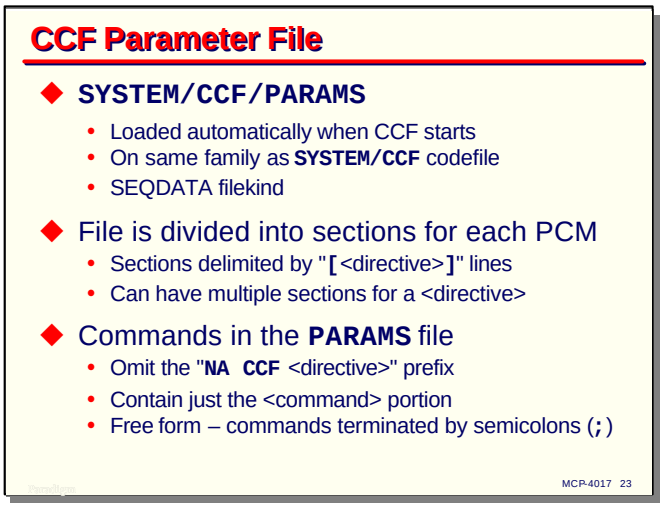
All CCF OI commands have the general form

NA CCF <directive> <command>

The <directive> portion of the command can be the name of a PCM (**CUCIPCM**, **TERMPCM**, etc.), the word "**ALL**", or the word "**ROUTER**". Any of these can be prefixed by the word "**TOPRINTER**" which causes the output of the command to be written to a printer file rather than displayed to the originator.

The <directive> indicates to which module(s) the command will be sent. If the directive is a PCM name, the command is sent only to that PCM. If it is **ROUTER**, the command is sent to the CCF Router. If it is **ALL**, the command is sent to all CCF modules, including the Router.

The <command> portion of the command begins with a keyword and typically has parameters or attribute assignments that follow that keyword. The examples show a command that disables a TCPIPPCM port and one that inquires on the status of all CCF modules, sending the output to a printer file.

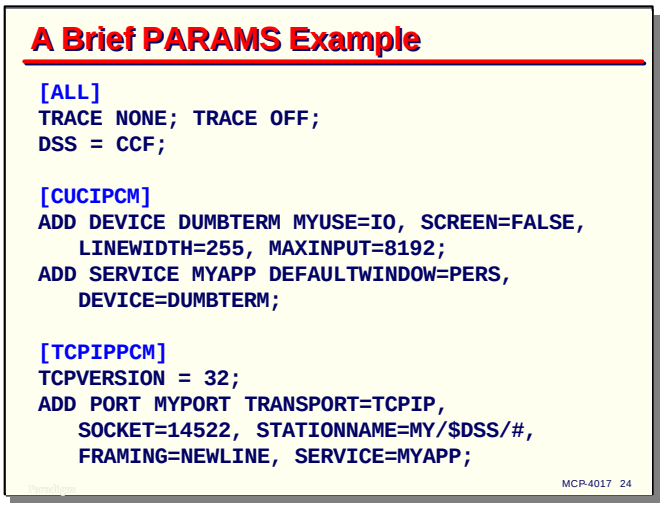


The text file that stores the initial CCF configuration is **SYSTEM/CCF/PARAMS**. This file is loaded automatically when CCF starts. It must be on the same family as the **SYSTEM/CCF** codefile. The file must be in CANDE SEQDATA format.

The file is divided into sections for the Router and each PCM. Each section is headed by a line with a <directive> in square brackets, e.g., "**[ROUTER]**" or "**[TERMPCM]**".

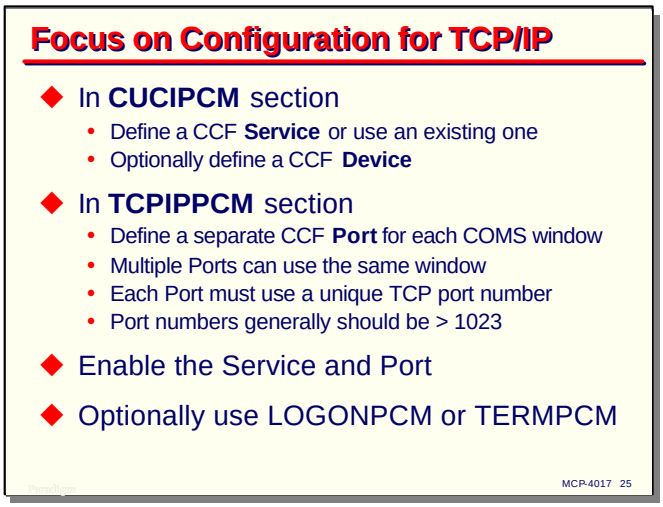
All of the commands for a given PCM do not have to be in the same part of the file. You can have multiple sections for a <directive>. This makes it easy to modify the configuration – you can construct a small file with just the changes you need (including the bracketed header lines) and either append it to the PARAMS file or use it with the CCF **LOAD** command.

Within the PARAMS file, the commands omit the **NA CCF** <directive> prefix, since that is implied by the previous header line, and contain just the <command> portion. The <command> is free form. It can be split across lines as necessary. It must be terminated with a semicolon.



Here is a brief example of a CCF PARAMS file. Note the header lines with the square brackets. Also note that the individual commands start with things like **ADD** and **TRACE**. These are just the <command> portion of CCF OI commands.

Once again, these header lines can be repeated in the file. The commands for a particular PCM do not have to be all grouped together in one section.



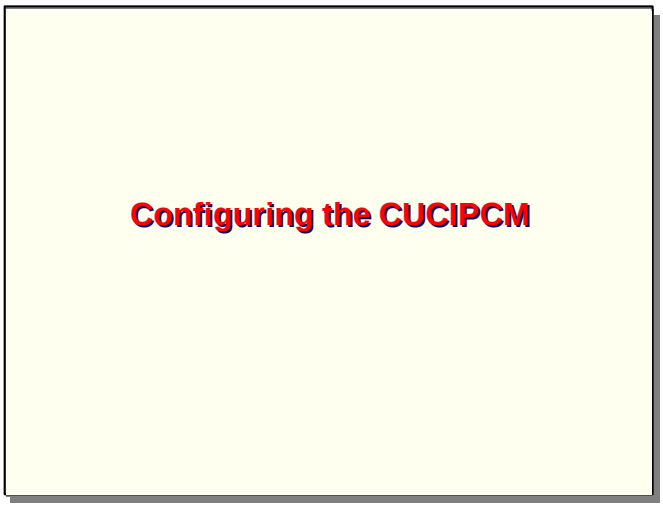
In the remainder of this presentation, I will focus on basic configuration for TCP/IP connections. This generally involves configuration for two PCMs.

In the CUCIPCM section, you must define a service or use one of the existing ones. There is no limit to the number of connections which can be routed to an individual service. You may optionally define a device. This is frequently not necessary, as some useful ones are included in the default configuration.

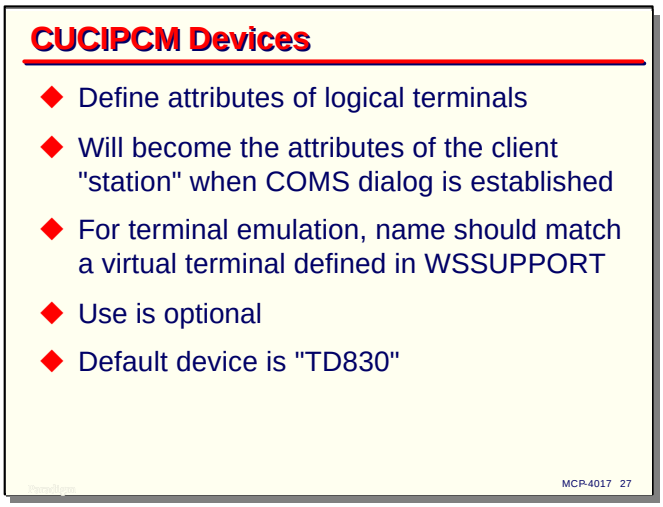
In the TCPIPPCM section, you must define at least one port for each COMS window you need to access. Multiple ports can be routed to the same window, but a given port can address only one window. Each port definition must also be associated with a unique TCP port number. This uniqueness must be system-wide, not just within CCF (or if the port is associated with a specific local IP address, unique within that IP address). You should choose port numbers which you know are not being used by other applications within the MCP environment. Normally these port numbers should be greater than 1023, since the lower numbers are reserved for the so-called "well-known" ports – Telnet, FTP, etc.

Once you have the CUCIPCM service and TCPIPPCM port defined, you must enable them so they can be used. This is normally done within the PARAMS file, but could be done via the OI at run time.

Optionally you may wish to use the LOGONPCM or TERMPCM. For most client-server or server-to-server applications, though, these are unnecessary.

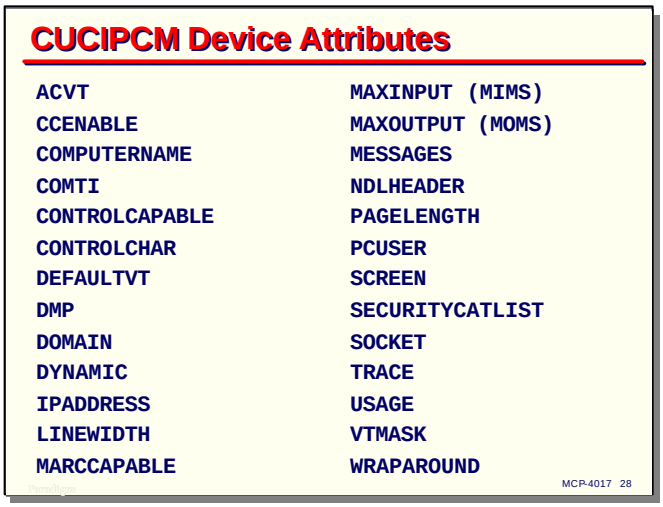


Let us now examine what you do to configure the elements of the CUCIPCM.



First, let us talk about configuring devices. CUCIPCM devices define the attributes of logical terminals. These attributes will become those of the client logical station when a COMS dialog is established for a connection using this device. For terminal emulation applications, the name of the device should match that of a virtual terminal defined in the site's WSSUPPORT library. WSSUPPORT provides message editing similar to that which was originally performed by NDL and NDLII in the datacom front-end processors.

The use of devices is entirely optional. If a device is not specified for a connection, CCF will use the "**TD830**" device as the default.



This slide shows all of the CUCIPCM attributes for devices. I won't go through these in detail here; they are described in the CCF Help file.

Most of these attributes describe physical characteristics of a terminal, and a some relate to COMS security settings. A few are worth mentioning, however.

ACVT (actual virtual terminal) specifies the virtual terminal CCF expects the client to be using. This cannot be overridden by the client.

COMTI (Boolean) specifies that the connection is to a COMTI (Microsoft Host Integration Server, or HIS) system.

COMPUTERNAME is the name of the remote client system.

DEFAULTVT is the name of the virtual terminal that is implemented by the client.

DMP (Boolean) specifies that the connection uses the Dialog Management Protocol.

DOMAIN is the DNS name of the client.

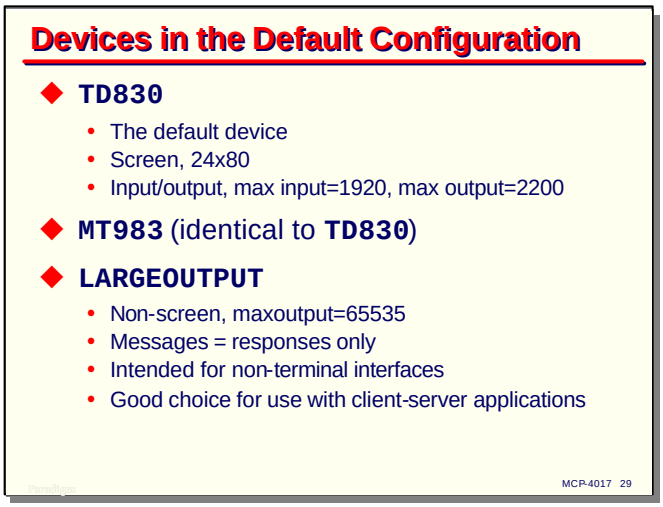
DYNAMIC (Boolean) specifies that this connection will be dynamic within COMS. After the connection terminates, COMS will automatically remove the corresponding station entity from its CFILE.

MESSAGES can have a mnemonic value of **ALL**, **NONE**, or **RESPONSES**. It controls what kinds of system messages will be sent to the client by COMS.

NDLHEADER (Boolean) specifies whether messages to and from the client will have a 6-byte NDL header attached.

PCUSER is the user name associated with the client.

VTMASK is intended to specify a list of virtual terminals the client can support, but presently is for information only.

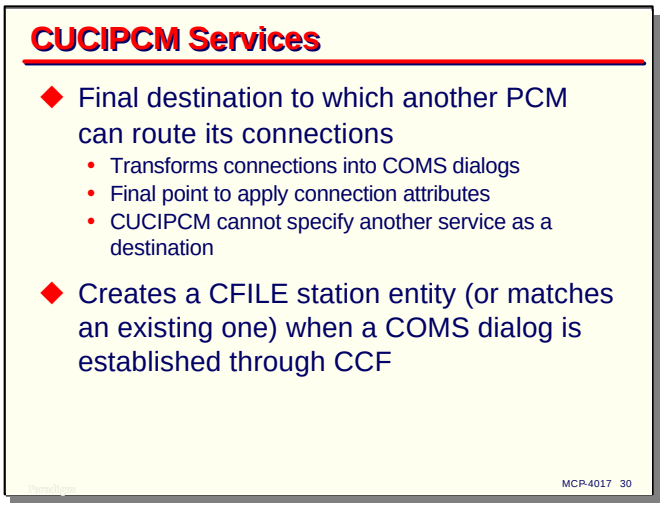


There are three devices defined in the default configuration. You are free to use these as you wish.

TD830 is the default. This is the device that will be used if no other is associated with a connection. It is specified as a screen device, 24 lines by 80 characters per line, input/output mode, with a maximum input (MIMS) of 1920 and a maximum output (MOMS) of 2200.

MT983 is a device with attributes identical to that of **TD830**.

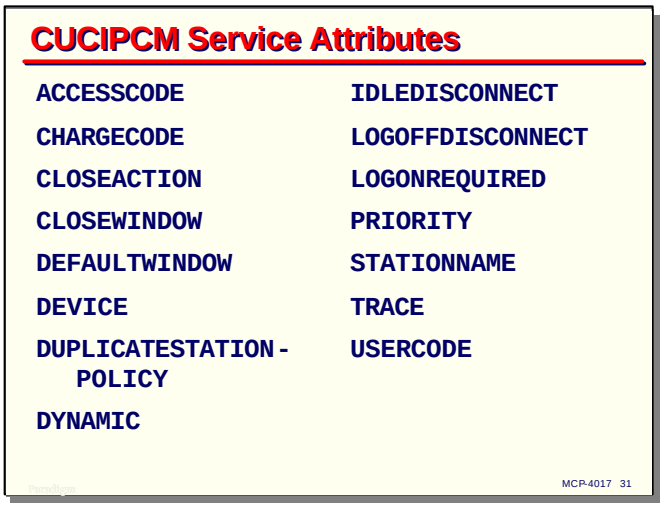
LARGEOUTPUT is defined as a non-screen device with a maximum output of 64K bytes. It also has the **MESSAGES** attribute set to **RESPONSES**. This device is a good choice for non-terminal interfaces, such as client-server or server-to-server.



The next topic in configuration of the CUCIPCM is services.

As mentioned earlier, CCF services are destinations to which other PCMs can route connections. In the case of the CUCIPCM, services are the last destination in the chain. As such, the CUCIPCM services are the only ones which cannot specify another service. CUCIPCM is the interface to COMS, so this is the final point at which CCF can assign attributes to a connection.

CUCIPCM transforms the connection into a COMS station dialog. Therefore, the attributes associated with a connection after being transformed by this PCM will either create a station entity in the COMS CFILE or (if the station names match) be applied to an existing COMS station entity.



This slide shows all of the connection attributes associated with a CUCIPCM service. Most of these are the same as the attributes for a COMS station entity. Two, however deserve special mention.

PRIORITY is an integer in the range 0-255. It is possible for services in CCF to have duplicate names. If a PCM routes a connection to a service name that occurs more than once in the configuration, the CCF Router will select the service with the highest **PRIORITY** value.

DUPLICATESTATIONPOLICY defines which connection attributes constitute a duplicate COMS station. If these attribute values for a new connection match an existing dialog, it's considered to be a duplicate. When a duplicate station is detected under this situation, the *original* connection will be:

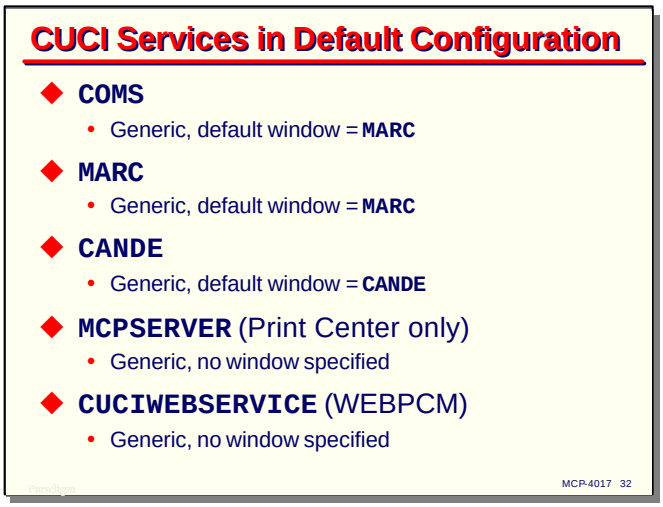
- **Parked** if it is less than 60 seconds old
- **Closed** if it is older.

This attribute is specified as a "displace list," for example:

```
DUPLICATESTATIONPOLICY = DISPLACE IPADDRESS + SOCKET,
```

The following attributes can be used in the displace list.

- **ACCESSCODE**
- **CHARGECODE**
- **COMPUTERNAME**
- **DEFAULTWINDOW**
- **DEVICE**
- **DOMAIN**
- **IPADDRESS**
- **PCUSER**
- **SOCKET**
- **USERCODE**



The default configuration has a number of predefined services. You can use these or create others of your own.

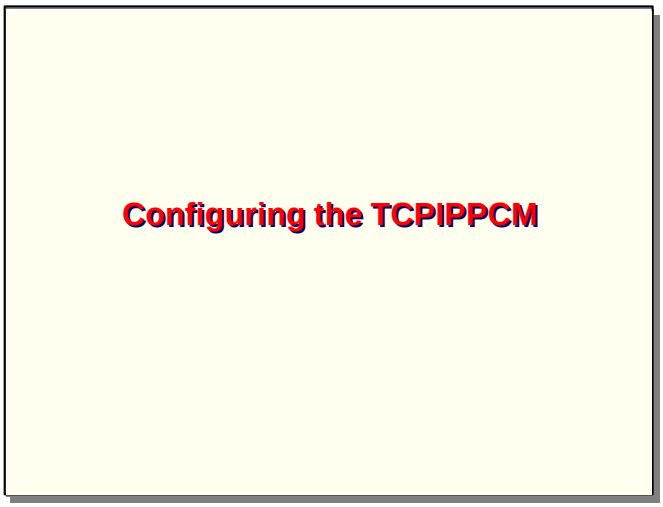
COMS is a completely generic service. It has a default window of MARC, but this can be overridden by an earlier PCM in the connection path.

MARC is similar to COMS.

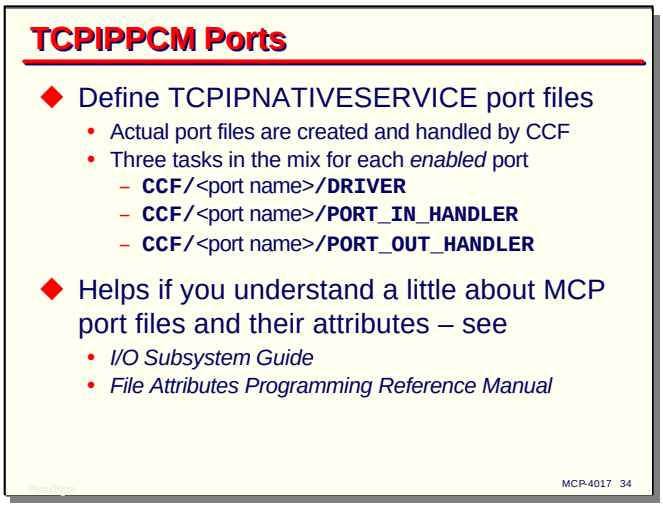
CANDE is similar to both of these, but has a default window of CANDE.

MCPSERVER is used by Print Center.

CUCIWEBSERVICE is typically used by WEBPCM services, but could be used by others as well.

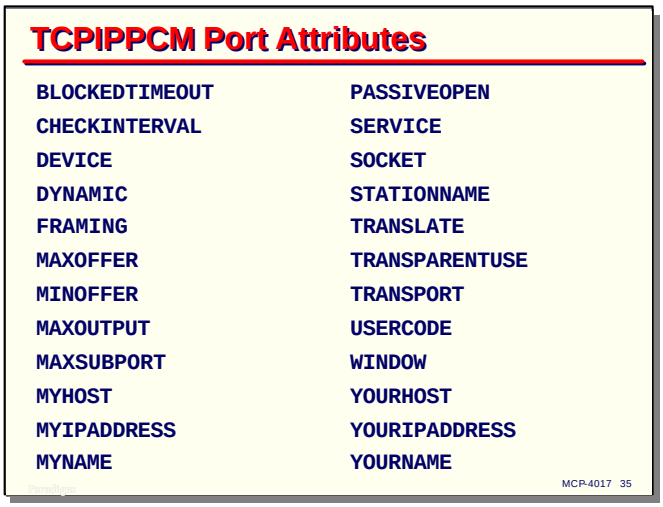


Having configured the CUCIPCM, let us now discuss configuration of the TCPIPPCM



The primary activity in configuring the TCPIPPCM is the definition of "ports." A port corresponds to a TCPIPNATIVESERVICE port file. The TCPIPPCM implementation creates and manages these port files at run time. It fires off three tasks in the mix for each enabled port: a driver task, an input handler, and an output handler. A single port definition can potentially handle up to 64K connections. Regardless of the number of connections, just three tasks per port are in the mix. Disabling a port will cause these three tasks to terminate.

Because the TCPIPPCM uses standard port files, it helps if you understand a little about MCP port files and the attributes used with the TCPIPNATIVESERVICE version of them. These are discussed in detail in the *I/O Subsystem Guide* and the *File Attributes Programming Reference Manual.*



This slide shows the connection attributes associated with TCPIPPCM ports.

DEVICE specifies a CUCIPCM device name to be used with the connection.

DYNAMIC (Boolean) specifies whether the connection will generate a temporary entry in the COMS station table.

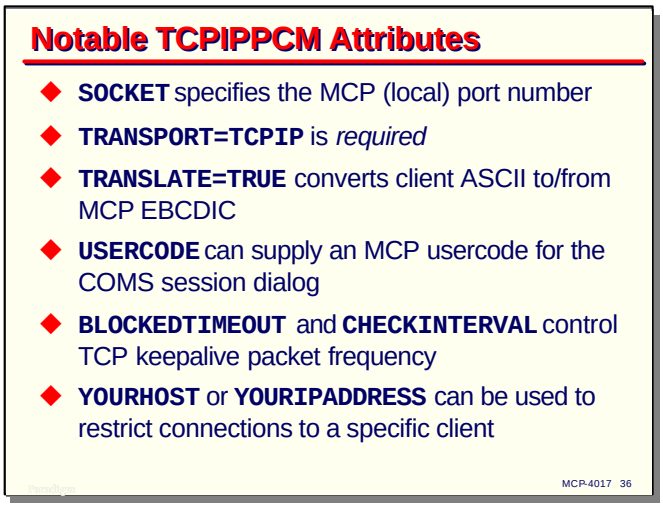
MYHOST and **MYIPADDRESS** can be used to specify which TCP/IP interface on the local MCP system will be used for the connection. Without this, the connection can be established through an arbitrary local interface.

MYNAME specifies the local TCP port number; it is the same as **SOCKET**.

SERVICE specifies the name of the next CCF service in the connection path. This will typically be the name of a TERMPCM or CUCIPCM service.

WINDOW specifies the name of the initial COMS window to which the station for this connection will be assigned. This may override any default window specification for PCM services later in the connection path.

The remaining attributes are discussed in the following slides.



There are several port attributes that deserve special mention.

SOCKET specifies the local (MCP-side) port number for the TCP/IP connection. This should normally be a number in the range 1024-65535. This attribute is always required.

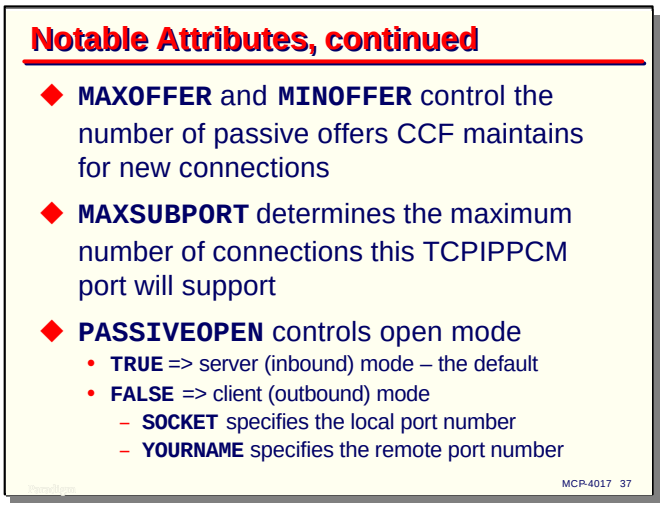
TRANSPORT=TCPIP is also required. It specifies that CCF should use TCPIPNATIVESERVICE port files. The reason this must be specified is that on earlier releases there were other types of port files, particularly TCPNATIVESERVICE.

TRANSLATE (Boolean) specifies whether CCF should translate the data stream from EBCDIC to ASCII when sending and from ASCII to EBCDIC when receiving.

USERCODE can be used to specify an MCP usercode for the resulting COMS dialog. Without this, the COMS session must log on using some other method, say via the LOGONPCM, a default usercode associated with the COMS station entity, or an explicit MARC log-on sequence by the user once the connection is established.

BLOCKEDTIMEOUT and **CHECKINTERVAL** correspond to port file attributes that control the frequency and duration of TCP "keepalive" packets. Both are specified in minutes and range from 0-1440. **CHECKINTERVAL** indicates how long the MCP should wait before starting to send keepalives on a non-responsive dialog. **BLOCKEDTIMEOUT** indicates how long the keepalives should continue to be sent before the connection either becomes responsive again or the MCP places it a "blocked" state.

YOURHOST or **YOURIPADDRESS** optionally can be used to define the network address of the remote system. If these are not specified, then inbound connections are accepted from any network address. If specified, inbound connections are accepted only from the specified domain name or IP address. This can be useful as a component of security in server-to-server applications.



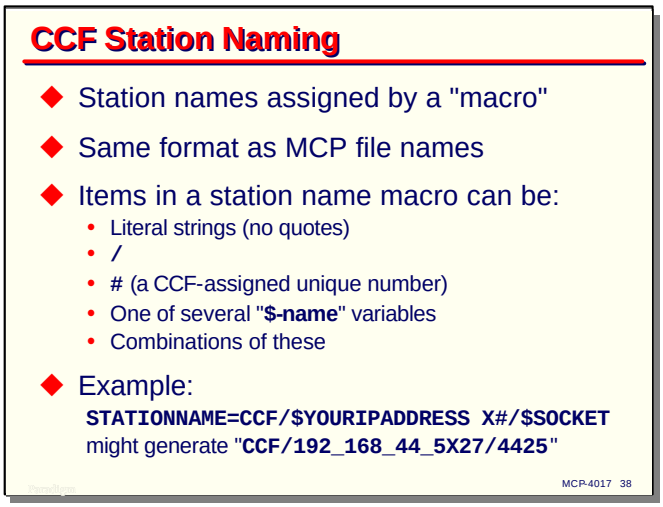
Continuing with some TCPIPPCM connection attributes that deserve special mention:

MAXOFFER and **MINOFFER** control the number of passive offers (port subfiles awaiting connections) that CCF maintains for new connections. **MAXOFFER** specifies the total number of passive offers that will be available at any one time. **MINOFFER** specifies how many of these passive offers will be offered at a time. **MINOFFER** is usually set to a relatively low number when **MAXOFFER** is large; this allows CCF to ramp up the number of offers gradually rather than dumping them on the network all at once.

MAXSUBPORT specifies the maximum number of active connections that this TCPIPPCM port will support at a time. It corresponds to the port file attribute of the same name. This must have a value in the range 0-65535.

PASSIVEOPEN controls the mode by which the subports are opened. This is not well documented, but I have determined the following by experiment:

- **PASSIVEOPEN=TRUE** is the default, and is normally not specified. It indicates that the MCP is acting as a server and is making passive offers available to the network for clients to connect. These are sometimes called "inbound" connections, since the connection must be initiated by the client. The **SOCKET** attribute specifies the port number through which the clients connect.
- **PASSIVEOPEN=FALSE** indicates that the MCP is acting as a client and that it will initiate the connection. **SOCKET** specifies the port number that will be used on the MCP side of the connection (apparently this must be specified, which is unusual for the client side of a TCP connection). The **YOURNAME** attribute specifies the TCP port number on the remote (server) side of the connection. **YOURHOST** or **YOURIPADDRESS** should be used to specify the address of the remote system.

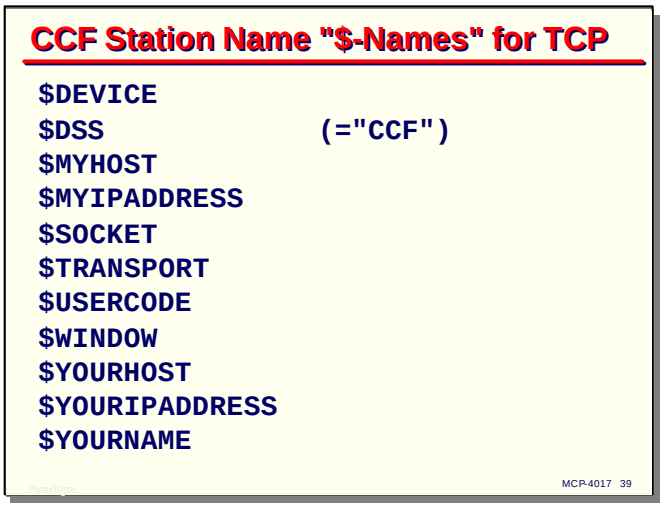


Since CCF ultimately transforms network connections into COMS station dialogs, COMS will need to have unique station names for those dialogs. CCF provides a "macro" mechanism to generate these names. Station names have the same format as an MCP file name – up to 14 nodes of 17 characters each.

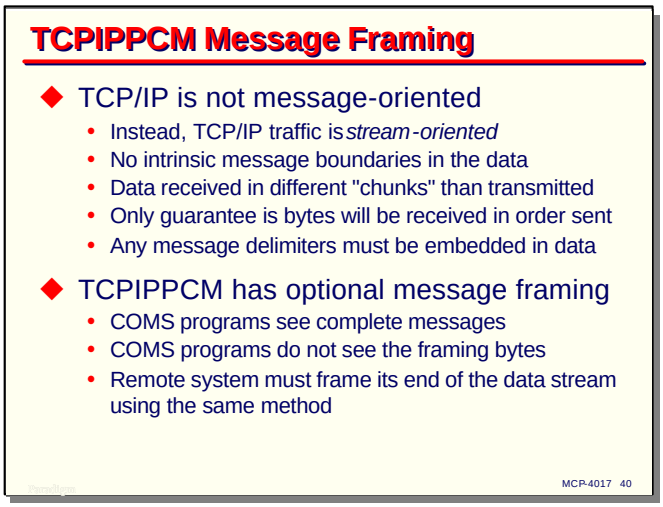
Elements of a station name macro can consist of

- Literal strings, without enclosing quotes.
- The "**/**" character to separate nodes of the name.
- The "**#**" character, which CCF will replace with an integer that is guaranteed to make the name unique.
- One of several "$-name" variables which CCF will replace with information for a specific connection.

As the example shows, station names using this macro would have names beginning with "**CCF**", followed by a "**/**", followed by the remote system's IP address, followed by the letter "**X**", followed by a unique number, followed by another "**/**", and finally followed by the local socket (port) number for the connection.



This slide shows all of the "$-name" variables which can be used with station name macros in the TCPIPPCM. Most PCMs (including CUCIPCM) can specify a station name macro, and any conflicts in name assignment along the connection path through CCF are resolved in the normal manner. Different PCMs have somewhat different sets of "$-name" variables, so the possibilities for naming a station can vary depending on which PCM actually supplies the name.

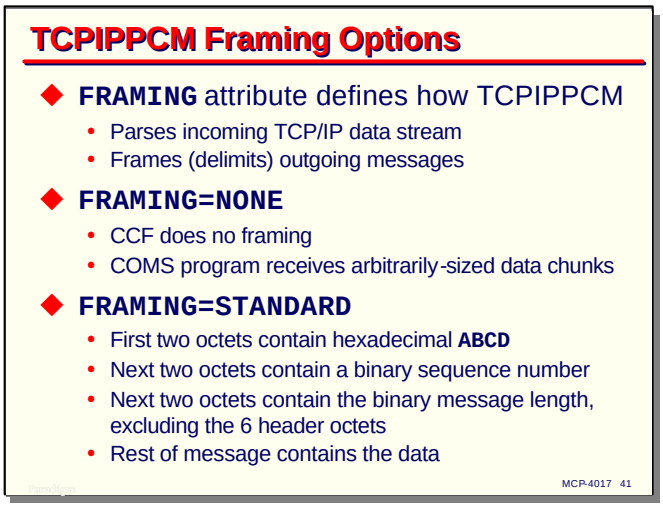


Another capability of the TCPIPPCM, which is unique to it, is that of message framing. Most of us who have worked with communications on the MCP are accustomed to the idea that a "message" is a well-defined entity, which is preserved as it is transmitted across a network. In other words, when one end sends a specific string of bytes as a message, the other end should receive that specific string of bytes as a message.

TCP/IP data transmission, however, is not message-oriented. Instead, it is *steam-oriented.* Data is transmitted and received simply as a stream of bytes – there are no intrinsic boundaries that delimit what we would call messages. This means that the receiving end may get the data in different-sized chunks than the sending end put on the wire. The only guarantee that TCP/IP makes is that the receiving end will get all of the bytes sent, in the order they were sent. Any delimiting of the data stream to form messages is a matter of convention between higher-level software on the two ends.

Since COMS is very much a message-oriented package, CCF provides several methods to "frame" messages in the data stream. This allows COMS programs to see complete messages without worrying about generating and parsing delimiters in the data. The insertion and removal of framing bytes is handled entirely by the TCPIPPCM – COMS programs see only the message data, not the framing bytes.

Since message framing is a matter of convention between the two ends of a TCP/IP connection, the remote system must, of course, use the same framing method as is specified in CCF in order for the messages to be properly recognized by both ends.



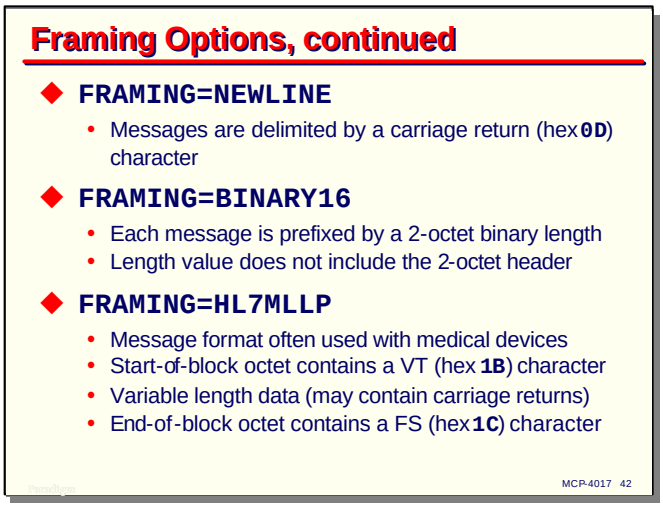
The TCPIPPCM **FRAMING** attribute defines the framing method which will be applied to a connection. This will control how the TCPIPPCM parses the incoming TCP/IP data stream to extract whole messages for COMS, and how it will add framing characters to the outgoing data stream.

FRAMING=NONE. The most basic form of framing is none at all. CCF simply forwards chunks of data as they are received from the network. COMS programs will receive these arbitrarily-sized chunks. This method is typically used when the remote system uses a framing method that CCF does not support. In this case, the COMS TP is responsible for all parsing of input to detect message boundaries, assembly of message fragments, and delimiting of output.

FRAMING=STANDARD. This is a robust framing technique which is unique to CCF. This is a very good choice if you can control how messages are formatted and sent on the remote system. Messages are prefixed by a six-byte header, thus:

- The first two bytes contain the hexadecimal value "**ABCD**".
- The next two bytes contain a binary sequence number. This number increments by one for each message sent, and is used as a continuity check. It wraps from 65535 back to zero.
- The next two bytes contain the length of the message data in binary. This length does not include the six-byte header. Thus, messages using this framing method can be up to 65535 octets long.
- Following the six header bytes is the message data, for the length specified in the header.

The binary numbers in this header are all sent in so-called network order, or big-endian. The high-order byte is first, followed by the low-order byte.

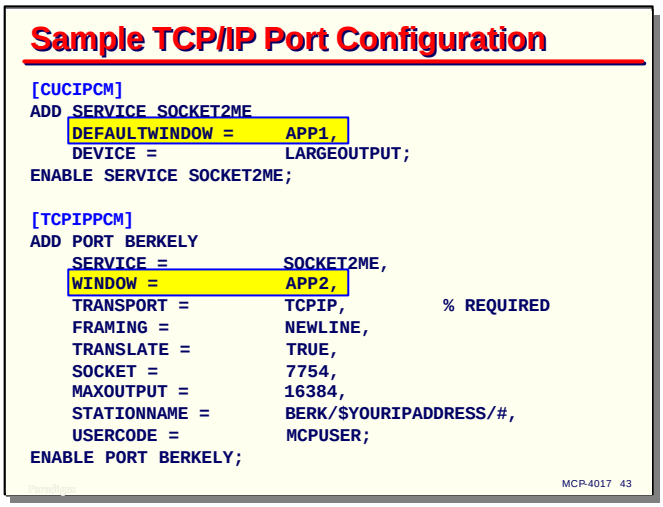


Continuing with the TCPIPPCM framing options:

FRAMING=NEWLINE. This framing method simply delimits messages with a carriage-return (ASCII hex 0D) character.

FRAMING=BINARY16. This framing method prefixes each message with a two-byte header containing the message length in binary. This method is frequently used by IBM systems and some ATM controllers. As with the **STANDARD** method, the binary message length value is sent in network order and does not include the size of the two-byte header.

FRAMING=HL7MLLP. This framing method, known as Health Level 7 Minimal Lower Layer Protocol, is commonly used with medical devices. It frames the message with an ASCII VT (hex 1B) byte on the front and an ASCII FS (hex 1C) byte at the end. The enclosed message data may contain only the printable ASCII characters and carriage-return.



This completes the configuration of TCPIPPCM ports. Putting this information together, here is a sample configuration file showing a TCPIPPCM port definition routing to a CUCIPCM service definition.

Note that the TCPIPPCM port specifies the name of the CUCIPCM service as a destination. It is using the **NEWLINE** framing method, and is translating the data stream to and from ASCII. Clients will connect to this port over TCP port number 7754. The resulting COMS stations will have names with three nodes, viz, "**BERK**", followed by the remote system's IP address, followed by a unique number. COMS sessions initiated through this port will run under the MCP usercode "**MCPUSER**". Any further client authentication will be the responsibility of the application.

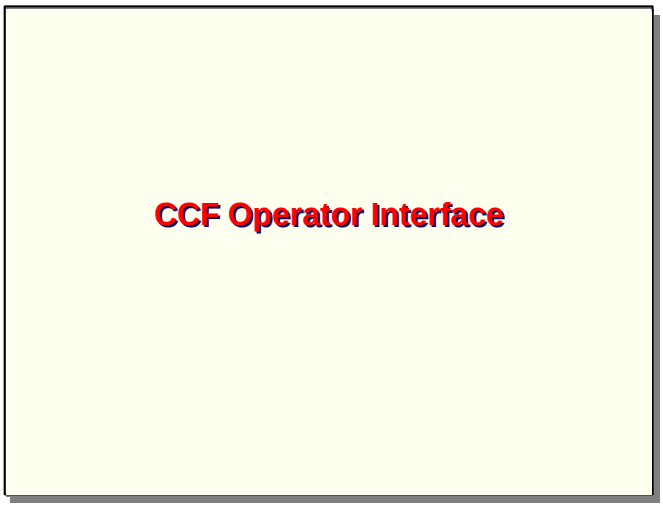
Also note that the two entities must be enabled before they can be used. The enable commands can be placed in the configuration file or entered manually after the configuration is established.

Also note that both the TCPIPPCM port and the CUCIPCM service define a window. Which one will be used? Since the TCPIPPCM port is closer to the client, its attribute settings take precedence, so these connections will be assigned to the COMS window **APP2**.

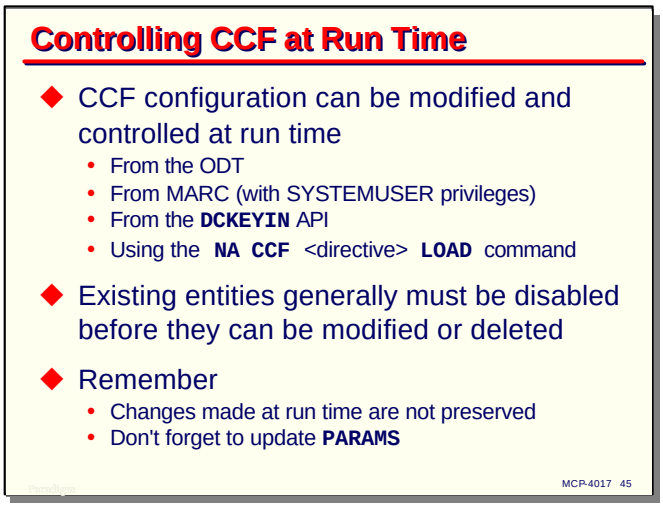
If the **WINDOW** attribute had not been specified for the port, or if the CUCIPCM service had specified

```
DEFAULTWINDOW = OVERRIDE APP1,
```

Then the CUCIPCM window specification would have been used, and connections would be assigned to COMS window **APP1**.



We have seen a portion of the CCF operator interface as used in the PARAMS file for configuring PCMs. This next section discusses the OI in more detail.



As I mentioned earlier, the CCF configuration is completely dynamic and can be modified and controlled at run time. The **NA CCF** commands can be entered from the ODT, from the Action line of a MARC screen (assuming the user has SYSTEMUSER privileges), and from system APIs such as **DCKEYIN**. You can also place one or more commands in a SEQDATA text file, using the same formatting conventions as for the PARAMS file, and load this text file using the **NA CCF** <directive> **LOAD** command. If the text file will contain commands for more than one PCM, you can enter

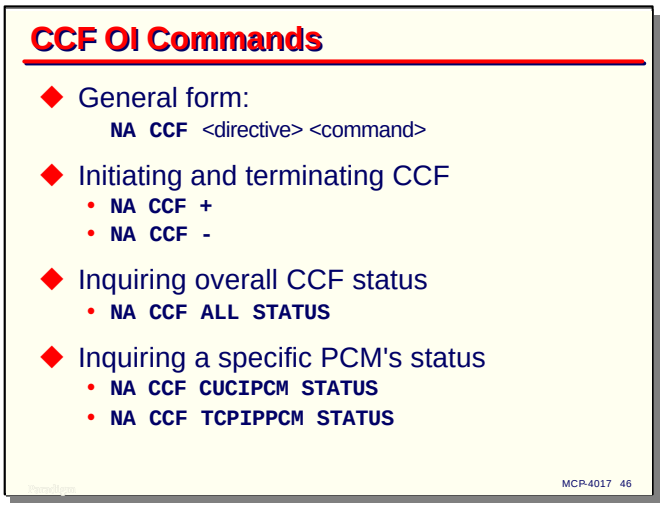
NA CCF ALL LOAD <file title>

to have commands for all of the PCMs processed at the same time. If you name a specific PCM for the <directive>, only the commands for that PCM will be processed from the file.

To modify or delete CCF entities, you must generally disable them first. Disabling a port or service causes it to cease working, and generally will terminate all connections using it. After modifying an entity, you enable it again to put it back in service.

Remember that any changes to the configuration you make at run time are only reflected in CCF's memory tables. These changes will be lost across a restart. Any changes that are to be made permanent must be included in the PARAMS file.

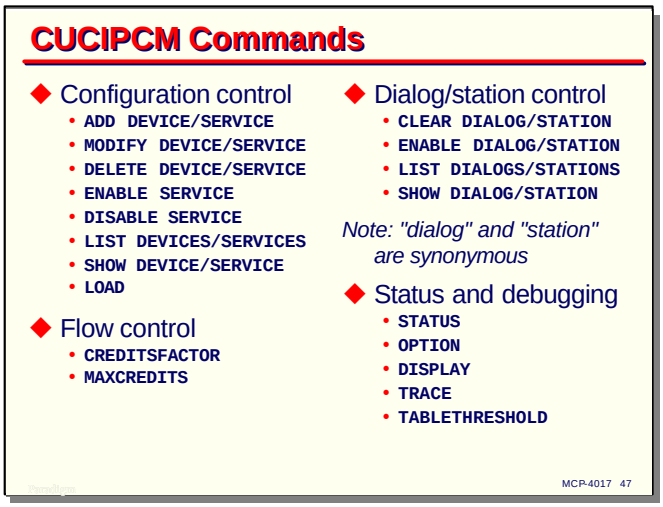
When adding new entities to the configuration, creating a text file with the related commands is a good idea. The **LOAD** command can then be used to add the entities, and once the new configuration is stable, this text file can be simply appended to the PARAMS file to make the changes permanent.



To review, CCF operator interface commands have the general form of "**NA CCF**" followed by a <directive>, which can be a PCM name, **ROUTER**, or the word **ALL**. Following that is the text of the command.

CCF as a whole can be started and stopped using the standard DSS convention. "**NA CCF +**" will initiate CCF if it is not running; "**NA CCF -**" will terminate CCF and all of its connections. This termination may take a couple of minutes to finish, as the connections must be closed first.

You can inquire on the overall status of CCF with one command, "**NA CCF ALL STATUS**". This will report the status of the Router and each PCM individually. You can also inquire as to a specific module's status.



This slide shows all of the OI commands for the CUCIPCM. We have already seen the **ADD** command in some detail. The **MODIFY** command is similar. Devices and services can be deleted. Services (but not devices) can be enabled or disabled. Services must be disabled before they can be modified or deleted.

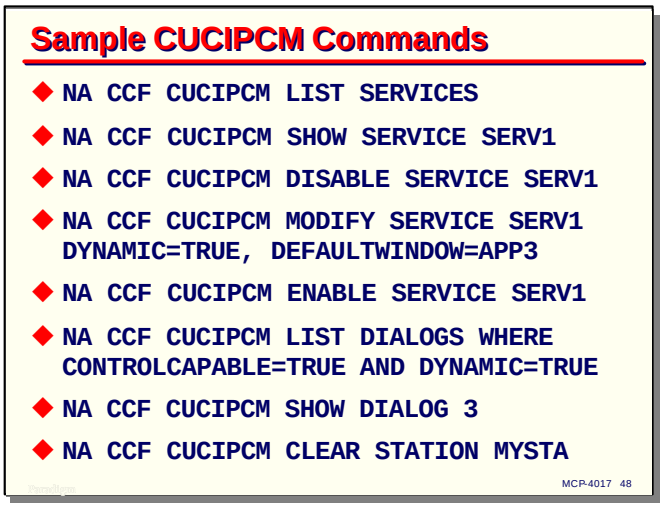
You can list the devices and services in the current configuration. This will show the names, a CCF-assigned number for each, and a brief summary of their status. The **SHOW** command will display detailed information about an entity's attributes.

As previously discussed, the **LOAD** command will load a text file of commands.

The **CREDITSFACTOR** and **MAXCREDITS** commands set flow-control parameters. The default settings are almost always adequate.

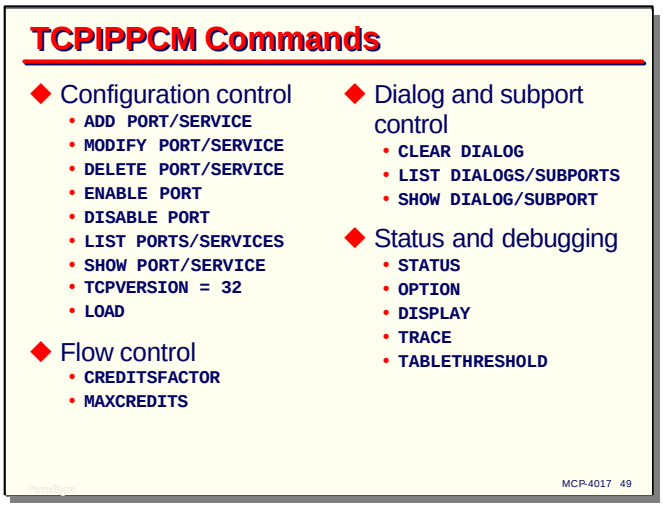
Since the CUCIPCM converts connections to COMS station dialogs, there are commands to interrogate and control these. The terms "dialog" and "station" are synonymous in these commands. The **CLEAR** command will terminate a COMS session and close the corresponding connection. The **ENABLE** command enables a currently inactive dialog. You can list the current dialogs and show the details of specific ones.

The **STATUS** command shows the status of the CUCIPCM module. **OPTION** sets various options for the module, but these seldom need changing. **DISPLAY** dumps portions of the CCF memory tables, and requires some knowledge of CCF internals to interpret. The **TRACE** command will record events to a printer file, and can be useful for diagnosing connection and configuration problems. **TABLETHRESHOLD** is used to control the sizing of the internal tables; again, the default value is almost always adequate.



This slide shows several samples of CUCIPCM commands as they might be entered from the ODT or a MARC screen. Note how the service is disabled before the **MODIFY** command and enabled afterwards.

One interesting capability of the **LIST**, **SHOW**, and **CLEAR** commands is that they can include a **WHERE** clause, which will display only the entities which match the list of attribute values specified in the clause. The **WHERE** attributes are inclusive, i.e., only the entities where all of the values match will be displayed.

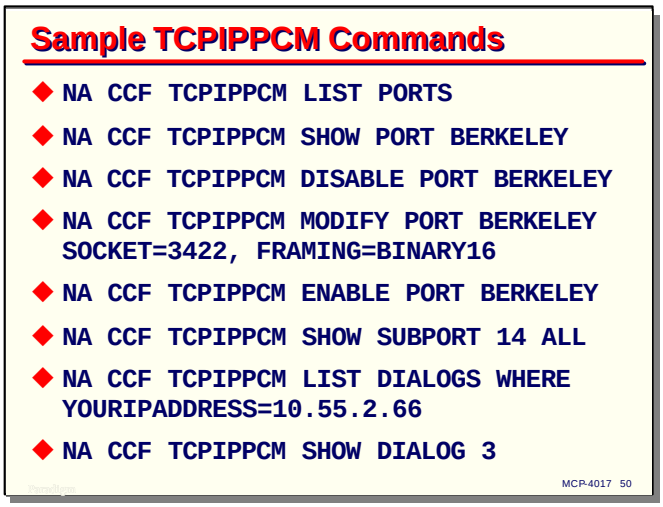


The OI commands for the TCPIPPCM configuration control are very similar to those for CUCIPCM, although the attributes used with **ADD** and **MODIFY** differ substantially between the two PCMs. The **TCPVERSION** command is typically placed in the PARAMS file and indicates that CCF is to use host-resident TCP/IP rather than the old CP2000-resident version.

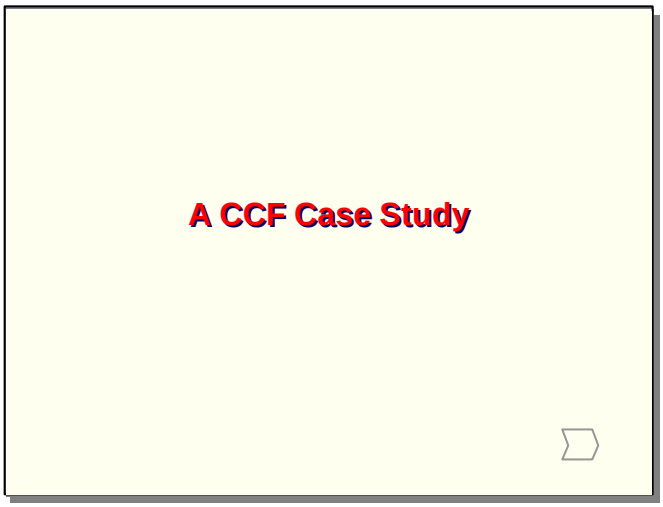
As with the CUCIPCM, you can list and clear the COMS station dia logs associated with a port.

Because the TCPIPPCM communicates on the network by means of port files, individual connections are managed through subports of those files. The **LIST** and **SHOW** commands also allow you to examine these subports.

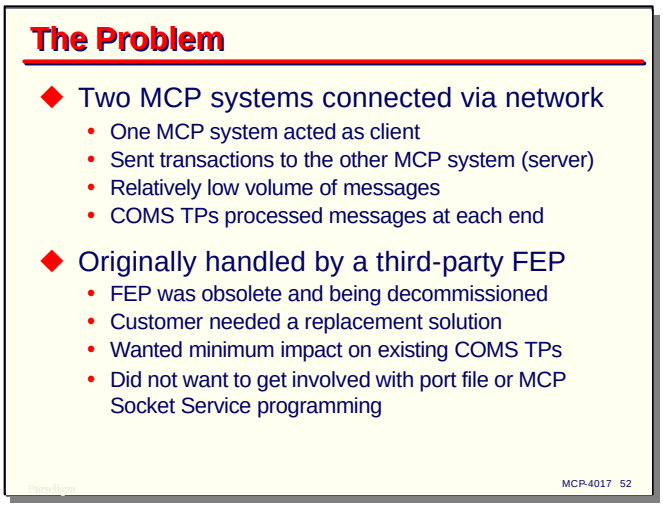
The flow-control, status, and debugging commands for the TCPIPPCM are essentially the same as for the CUCIPCM.



This slide shows several sample TCPIPPCM commands.

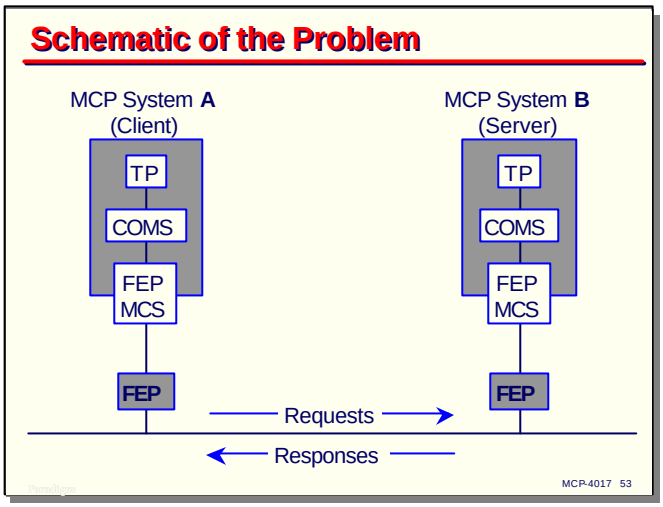


I had an interesting application for CCF not long ago and thought it would make a good example to review in this presentation.

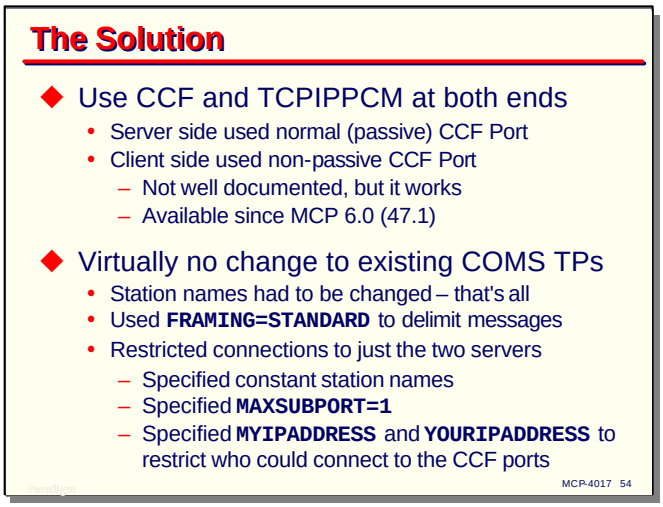


A customer contacted me with a problem. They have two ClearPath MCP systems located in different cities and connected through their private corporate network. They have an application that runs on these two systems, and the two sides need to communicate with each other. One system acts as a client and sends request transactions to the other, which acts as a server, processes the transactions, and returns responses to the client. The volume of messages is relatively low. They use COMS TPs to process the messages on both ends.

The link between the two systems was originally handled by a third-party FEP product. Each MCP system viewed the other as a normal datacom station. This FEP was obsolete and being decommissioned, and the customer need a replacement solution for the inter-system communication. They wanted to replace the FEP with as little impact as possible on the existing COMS TPs, did not use BNA, and definitely did not want to get involved with programming for port files or the MCP Socket Service.



This slide shows a schematic representation of the problem.

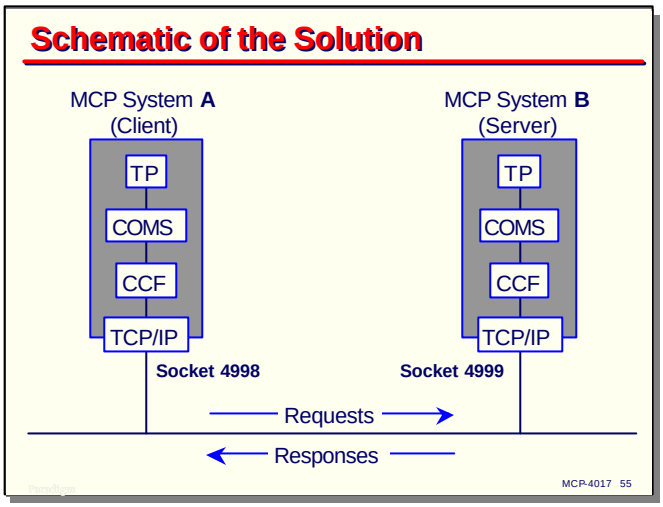


CCF to the rescue.

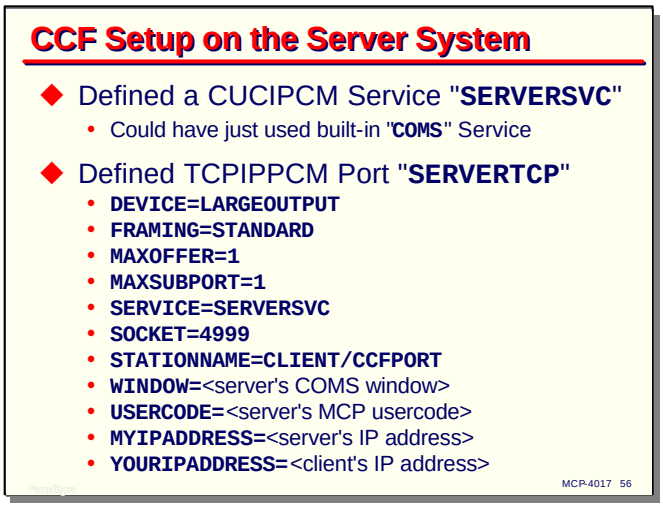
The solution was to use a TCPIPPCM port at both ends. The server side was configured in the normal fashion with a port that used passive open. The client side was configured with a port that use non-passive open. This use of the **PASSIVEOPEN** attribute is not well documented, and it required some experimentation to get it to work, but it does work. You need to be on at least MCP 6.0 (SSR 47.1) to use non-passive ports, as that is when they were first implemented.

This proved to be a nearly ideal solution, and except for several hours spent in experimenting with the configuration for a non-passive port, was quite easy to implement. In the end, no changes were necessary to the COMS TPs, except to the names of the COMS stations they used. Even this change was not strictly necessary, except that the customer wanted to leave the old FEP capability in place for a while after switching to the CCF implementation.

I set up the TCPIPPCM ports with **FRAMING=STANDARD**, since that is the most robust of the methods implemented in CCF. Since this was a one-to-one connection between known hosts, I also configured the ports with constant station names, **MAXSUBPORT=1**, and values for **MYIPADDRESS** and **YOURIPADDRESS** on both ends. This restricted the use of the interface to just those two systems. Because both were MCP systems, no translation between EBCDIC and ASCII was necessary.



This diagram shows a schematic of the solution. The client system uses TCP port number 4998 and the server system uses port 4999.

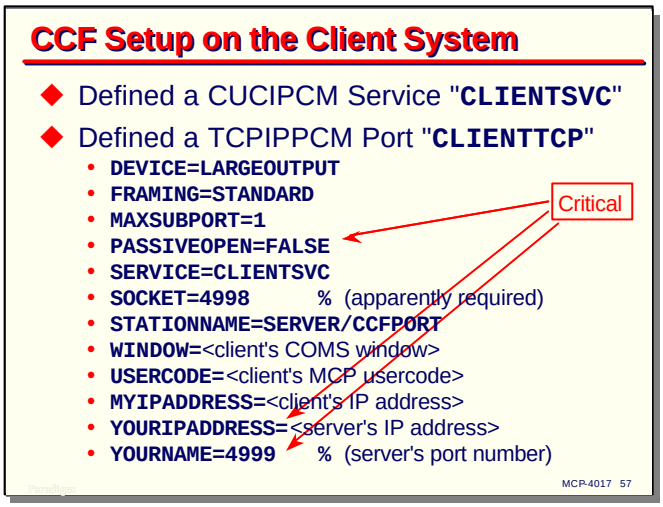


This slide summarizes the configuration on the server system. I defined a separate CUCIPCM service for the application to use, but could have just used one of the ones in the default configuration, such as the **COMS** service.

The critical part of this configuration is **SOCKET=4999**. This is the port number the client system will have to specify.

DEVICE=LARGEOUTPUT allows the messages to be sent between the two systems without any virtual terminal editing. Since there is only one connection, **MAXOFFER=1** and **MAXSUBPORT=1** specify this.

MYIPADDRESS and **YOURIPADDRESS** are not strictly necessary. **MYIPADDRESS** specifies which local TCP/IP interface (a shared adapter in this particular case) CCF will use. **YOURIPADDRESS** restricts connections to those coming from that specific IP address.

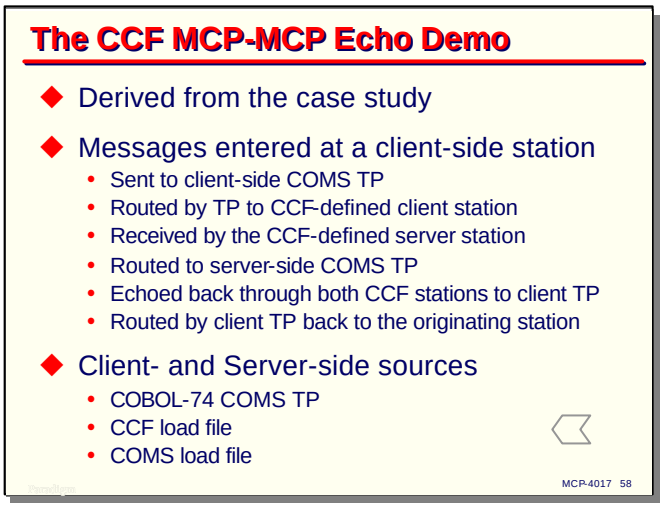


This slide summarizes the configuration on the client system. As on the server, I defined a CUCIPCM service, but this was not really necessary.

The TCPIPPCM configuration is very similar to that on the server, with four exceptions:

- **PASSIVEOPEN=FALSE**. This causes CCF to initiate a connection to the server system rather than sit and wait for one to come in.
- **YOURNAME=4999**. Since this is the active side of the connection, it must specify which port on the server it wants to connect to.
- **YOURIPADDRESS=**<server's IP address>. Also since this is the active side of the connection, it must specify the host to which it will communicate.
- **SOCKET=4998**. Normally when opening an active connection, you do not specify the local TCP port number, but let the operating system assign it from the pool of available numbers. CCF, however, apparently requires the **SOCKET** attribute to be specified for TCPIPPCM ports regardless of the setting for **PASSIVEOPEN**. The actual port number used here is not important, as long as it differs from any other local port number on that IP address.

Once again, **MYIPADDRESS** is not required. It simply restricts the connection to a specific local TCP/IP interface.



As an illustration of applying CCF to TCP/IP communications, I took the test bed I developed for the problem just discussed and packaged it as a demonstration.

On the client side, there is a COMS window and corresponding TP that will receive a message from a local station and route it to the TCPIPPCM station defined in CCF. That will cause the message to be sent out across the TCP/IP network to the server system, where CCF will route it to a COMS window and its TP. The server-side TP simply echoes the message back to the client system, where COMS routes it back to the client TP, which in turn returns the echo message to the original local station.

This demo consists of source files for the client and server TPs, along with text files for the COMS and CCF configuration that is necessary on both ends. These source files are available from our web site at the URL on the References slide, next.

References

- *Custom Connect Facility Administration and Programming Guide* (4310 3266-006)
  - Windows Help file on documentation CD
  - Unchanged since MCP 8.1 release
- Integration Expert (PathMate)
- This presentation and the MCP-MCP demo
  http://www.digm.com/UNITE/2005

End of
Using CCF

2005 UNITE Conference

Session MCP-4017