General Overview of the Component

MindMap consists of a set of basic building blocks - the components. These components have attributes which the developers can set to certain conditions. The fundamental building blocks are then linked to one another via events and messages.

The client/server component expands this metaphor to now include higher level building blocks - subassemblies. A developer can use the basic components to assemble a functional entity. This entity can then be used in other situations as though it were a basic building block.

In chemistry, this would be comparable to arranging elements to form molecules. In the electronic industry this is analogous to combining chips to form an adapter board which plugs into the motherboard.

It is important to note that this approach allows the inner working of the subassembly to be masked from the outer world. The subassembly presents itself to the rest of the world via a well-defined interface. Molecules have docking stations at which other elements or molecules can attach themselves. The interface between the motherboard and adapters is the bus (*ISA, EISA, PCI, etc.*).

In the MindMap environment, a special component - the client/server component - functions as the sole interface between the subassembly and the other components of an application. Thus, a c/s component encapsulates the complexities of the subassembly.
This construct leads to a number of advantages:

1. *Teams can develop applications with minimum interaction necessary between the groups.*
2. *A large application can be broken up into more manageable units (subroutines) and these subassemblies can be reused from within other applications.*
3. *Applications can initially be built locally and then deployed remotely, or they can actually be built at remote sites.*

In the case of **team development,** it is possible to have various individuals, or groups of individuals, work more or less independently of one another on the completion of an application. In this case, each individual would be assigned a specific task which would be realized in the form of such a subassembly. Each subassembly would make itself accessible via its encapsulator (c/s) component. Members would only have to agree on the functionality required from a subassembly and how it would be accessed.

When a large application is to be developed, it often becomes necessary to decompose the complexity by defining blocks of functionality. In traditional programming environments, this was done by means of **subroutines** and procedures. In the case of MindMap, the functional entities are assembled into a collection of basic building blocks and subsequently wrapped up in a subassembly. The c/s component (*encapsulator*) becomes the point of interaction between the subassembly and the rest of the world. If done properly, meaning with sufficient abstraction, this subassembly can be used in other applications to perform the same task. This is what was referred to as code reusability in the classical programming environments. In the MindMap context, it is called functional reusability.

If you are thinking in terms of disparate computers, then MindMaps client/server technology can be used to build **highly distributed applications**. A subassembly can be built so that it will run on a remote computer. If and when the functionality embedded in the subassembly is needed, a connection is established and the functionality is accessed. The construction of such distributed types of applications can be undertaken locally on one and the same machine and then deployed when complete, or can actually be done on the dislocated machines and merely connected for testing and deployment purposes.

In conjunction with the Internet or company-wide intranets, applications or portions thereof can be posted on easily accessible servers and downloaded at the time of execution. The machine which is to execute the application has the MindMap environment installed and only needs to download the application part, which in general is very small and can thus be easily transported. This makes for easy maintenance of central applications.

When two subassemblies are connected, the channel running between the two entities can transport two different types of information:

links; i.e., events and/or messages

data.

Links are passed just as they would be between two components in any other MindMap application. Data is passed in a seemingly somewhat more limited manner.

If component A wishes to send a message to component B, it actually has to send it to encapsulator component A. This component in turn receives an event, to which it then creates a message. This message is then sent to encapsulator component B. This encapsulator receives the event, generates a message and sends it to the component B. Passing data between two components residing in different subassemblies functions by the same token. Instead of sending data directly to the other component, it is forwarded to the intermediary agent - the encapsulator component.

In both cases - exchanging events and messages and passing data - the actual names of the events/messages and data do not have to correspond to one another. The developer of one subassembly - by definition - does not have any control over any of the aspects of the other subassembly. This in fact would actually defeat the purpose of encapsulation, were the developer of a subassembly to rely on his/her knowledge of the inner workings of any other subassembly.

At first, this might seem overly complex, but with a little practice it will become as easy to use as is the simpler relationship between conventional components. It is by communicating with an agent, instead of directly with the receiving component, that is the basis for the distribution of application functionality and for the encapsulation process.

Local Server

The process of building applications based on subassemblies is always the same, independent of where the various subassemblies reside - locally or remotely. In contrast to remote subassemblies, the facility to access local ones is installed by default. This is due to the fact that no specific communication channels have to be established. The computers on which remote subassemblies (*server*) reside must be installed in MindMap. This process is described in detail in the next segment.

This default setting allows the immediate construction of applications based on subassemblies.

Installing Remote Servers

If you wish to access facilities residing on remote systems, you must first register the host system it resides on. To do this, select the menu option File | Preferences. Scroll down to the option Network.Next click on the **New** button and continue to install a new host.
It is easiest to have MindMap search for its communication drivers automatically. To initiate this process, click on the **Search Drivers** button. MindMap will search its working directory and will return all drivers it is able to locate. If you wish to manually locate the drivers, click the **Load Driver…** button. You will be displayed with a file selection dialog box. Navigate to the directory containing the appropriate driver and select it.

MindMap ships with two different communication drivers:

a TCP/IP connection (MMTCP.DLL),

a MS-Windows for Workgroup connection (MMWFW.DLL)

The first driver allows two systems to communicate via TCP/IP. This protocol is common on many LANs, as well as on the Internet.

The second driver can be used on a LAN which uses Microsofts Windows for Workgroups protocol.

Once MindMap has loaded the drivers, select the appropriate driver by clicking on it

At the top, you must enter a name by which you can address the host computer. This name is only used internally, so feel free to enter any name, as long as it fits into the field.

The second entry is more critical. It is the actual TCP/IP address by which your computer can access this host. You can either enter an actual TCP/IP address or its name. The latter is only a valid entry, when the host you are attempting to connect to has been registered with a name server (*DNS entry*). In this case you can enter the name under which the server has been registered, such as WWW.MYHOST.COM.

The third section of this dialog box will be dealt with, when we describe how your own computer can define itself as a MindMap server.

The fourth entry concerns the size of the packets to be transferred. By default this is set to 4096, since it is assumed that compression will be used. It is also the maximum packet size if compression is employed. If you choose to transmit data without compression, then the maximum setting is 64KB.

In some instances it might be necessary to be able to set additional parameters. These can be accessed by clicking on the **Advanced** button.

The first entry allows you to specify a port address. The second permits the entry of a timeout after which MindMap will disconnect.

Note:
The port settings on the client and on the server side must be identical, otherwise a transmission will not occur, since the data bypasses the transmitter and receiver respectively.

Once you have completed the entry of the required data, MindMap will register the server.

These have been the steps necessary to register a TCP/IP based remote server. In order to register a MS-Windows for Workgroups server, more or less the same steps are required.

Begin by selecting the **New** button. Then click on the **MS-Windows For Workgroups** option.

The first entry again is intended to enable you to identify the host computer. Enter any character string you feel allows you to easily recognize the host. This is the name by which the host computer will be listed in the selection list.

Next, you must specify the actual workstation name as it has been defined in MS-Windows for Workgroups. If you are not sure, press the **Search** button and MindMap will query the appropriate files and return a list of recognized workstations on the LAN. Assuming that you recognize the targeted workstation, select it and the name will be transferred to the field.

The packet size has its default setting of 256. Data compression is also selected by default.

Now the MS-Windows for Workgroup server has been registered and will appear in the list of available hosts.

You can register any number of servers, which will all appear in the list. As will be described in subsequent sections, MindMap can establish connections to any number of server over one and the same protocol. This means that multiple servers will share the same logical and physical connection. The TCP/IP and WfW connections are able to multiplex logical connections on top of physical connections.

Defining a Server Subassembly

The first step is to build a server instance of a subassembly. This server will be called by a client instance.

Note:
The terms Client and Server are actually misnomers. MindMap supports the concept of peer networks. Any subassembly can communicate with any other subassembly, given that both have an encapsulator component placed. One subassembly initially calls the other subassembly and is therefore referred to as the client, whereas the called subassembly is designated the server subassembly.

Begin by opening a new file. Select the icon representing the client/server component. Even before you can place it on the screen, a wizard takes control. This wizard will initially display two informational dialog boxes. Read through these, acknowledging each by clicking on the **Next** button.

As already mentioned, since this is the first client/server type application you are presumably assembling, you must select the first radio button. This allows the creation of the server part. Continue by clicking on the **Next** button.

This screen contains some additional information, just to remind you of the process of assembling client/server applications. Accept this wizard screen by clicking on the **OK** button. The cursor will change its appearance to reflect that an encapsulator component is to be placed. Move the mouse to the screen position at which you would like to place the component and click once on the left mouse button. MindMap will now place an indented square on the screen, representing the server instance.

This statement will be viewed by the calling client. It should express the actual function associated with it. In this case, no function is attached to it. In a real application, something like

> Initiate database query, or
>
> Retrieve video sequence

would be entered into this field.

Note:
The commands have been entered here with quotation marks. This has been done to make them easily distinguishable from the other text. Within MindMap, do not enter such characters - simply enter the command - without delineators.

After the first command has been entered, the next screen will appear.

This screen allows a selection regarding the direction of the connection. What this determines is who will be generating a message and who will be receiving an event. The default setting is the last button, reflecting a bi-directional flow.

Note:
In most real applications, the flow of the 'flow' is critical, so to say. It is strongly advised to give considerable forethought relating to the 'direction of the connection'.

For the time being, retain the default setting and click on the **Next** button.

Since we will not be entering additional commands at this time, please click on the **OK** button to exit the wizard.

Control will be returned to the MindMap screen and you can again see the indented box. It should contain the text **Server**.

If you havent already done so, please place it in the top left corner of the screen. Now press the screen size button of this little application. Dont **size** the general MindMap window - only the window associated with the little application. Now size the application window so that is approximately the size of the top left quarter of your visible screen area.

Now **save** it into the working directory of MindMap, calling it SRV1.MM.

Once this has been accomplished, close this application, but leaving MindMap still active. Open a new file, which we will turn into the client application.

Defining a Client Subassembly

This client application will do nothing else but call the server application SRV1.MM and subsequently close it again.

Begin by again selecting the encapsulator icon on the toolbox. As expected, the wizard will take control and present itself with the now familiar informational screens. You will be offered the following dialog box. Select the radio button associated with building a client component.

Leave this screen by clicking on the **Next** button.

The wizard automatically defaults to the local computer setting. Since we do have other servers registered, they appear in the list on the lower part of the dialog box. We will not be connecting to them, so retain the default setting and click on the **Next** button. This takes you to the next wizard screen.

MindMap will look for all potential server applications - meaning .MM files - residing in the MindMap working directory. These are displayed in the list. In addition, directories within the current working directory are listed and can be navigated to by double-clicking on them in the list.

Note:
You can change the home directory by accessing the File | Preferences | Network *option.*

Select the server application you just completed, highlighting it in the list. Finish this screen by clicking on the **Next** button again, or by double-clicking on the file name. You will view an informational screen which you should read and then acknowledge by clicking on the **OK** button. The control will return to the MindMap desktop and the appearance of the cursor will have changed to signal that you can now place the client instance of the encapsulator component.

Click once, anywhere on the screen, and the indented rectangle representing the component will be placed. Once that has been done, you might wish to select the component and increase its size somewhat to view some of the information displayed in it.

In principle, we have now established a connection between two MindMap applications. Lets save this client application and then proceed to make it actually interact with the server application. **Save it** under a file name such as CLNT1.MM, preferably into the MindMap working directory.

Assuming that the CLNT1.MM file is still open, lets create a command button to open the server application and one command button to close it again.

Place a command button on the desktop and call the linking facility. Use **Left mouse button released** as the event. Navigate in the message list until you locate **Client**. Select it…

The list in the left side of this dialog box contains all client instances of the encapsulator component contained in this application.

Note:
You can place as many client instances of the encapsulator component into an application as needed. In fact, you can also place a server instance into an application which contains client instances. A MindMap application can contain a maximum of ***one*** *server instance. A MindMap application can be called by, and can call, as many other subassemblies as are required. Thus, you can create networks of applications calling one another as needed.*

Here comes the solution to the riddle…

Once you select the client instance CLT1 in the list on the left, it will query the server instance, residing in the other SRV1.MM file, to determine its functionality. Since this file is located on the local machine, the process will take a fraction of a second. Were the file contained on a computer on the other side of the globe, then the designated communication path would be established (*i.e. TCP/IP via the Internet*) and the same process would be executed, obviously taking somewhat longer to complete.

The result of the interrogation will then be displayed in the list on the right side of the dialog box.

Three commands are available. The first two are generated by default and are always available. The third command is the one we created in the server application. It appears exactly as we had input it.

Since we want to open the server application, select the second command **<<<Open the Server>>>** and acknowledge the dialog box.

Now create a second command button and place more or less the same link on it, but instead of opening the server, generate the message **<<<Close the Server>>>**.

Again, save this little application and execute it. In run mode, click on the button you have designated to open the server application.

This is what the result should look like - again, more or less, depending on the labels you put on the buttons, and the dimensions you chose for the server window. To close the server application, click on the appropriate button in the client application.

Exchanging Events and Messages

Let us now take our little example one step further and have them interact with one another…

The first step will be to open the server application and place a rectangle in it. We will use the client to switch the fill color of the rectangle.

Open the server file SRV1.MM.

Place a rectangle in the top left corner of the screen. Then select the server instance and access its attribute toolbox. What we are looking to do is to add a new command to the server applications API (*application programming interface*).

Click on this icon and you will be presented with the following screen.

We will keep the dummy command and create a new one. Click on the **New** button.

Again, enter a text to represent the command. In this case we will use something of the sort:

Change my color to red

Leave the directional option set to the default of bi-directional, for the moment. Click on the **OK** button. Repeat this process to enter a command:

Change my color to blue

This was necessary to expose the functionality to the outside world. Now we need to define what the application will actually do when it receives either one of these commands.

Select the server instance and place a link on it. If you scroll down to the bottom of the event list, you will notice that three new events have been added to the list.

Select the event:

Change my color to red

When this event is received by the server instance, it is to generate a message for the rectangle to change its color to red. Do the same for the blue color change.

Return to the MindMap desktop, make sure that you have sized the screen so that it isnt in full screen before you save, and then save the application. Dont forget to save the application (SRV1.MM), since the client application will not know about the changes to the application, unless they have been saved. Now, close the file and load the client file CLT1.MM.

Begin the modifications to the client application by updating the connection between client and server. This is necessary to notify the client of new commands available at the server. In order to do this, select the client instance and activate its attribute toolbox. Only this time, select the icon for data exchange.

You will be presented with the following screen:
Click on the Refresh button and the client will call on the server once again to supply its updated API (*Application Programming Interface*).

Next, create a new command button which will be used to switch the color of the rectangle. Label it with an appropriate text. Continue by placing a link on the button which is to send a command to change the color to the server application - via the client instance and the server instance.

Save the application and execute it. If all went well, you should view a screen similar to this one…

Client Instance Attributes

The client instance of the encapsulator has a total of four icons on the attribute box. The **Linker** and the **Annotation** feature are common to all other components and thus will not be described here.
The **Redefinition of Instance** attribute allows you to redefine the properties of already placed instances. If the location of the corresponding server application has been altered, redefining is mandatory - but it leaves predefined links intact. This allows a server application to be built and tested locally on one machine and then moved to a remote machine without having to alter any underlying logic, other than reestablishing the connection (*so that the client knows where to go looking for the server*).

The **Data Exchange** attribute offers two different functions, albeit both are closely related.
The first function allows you to update the connection to the connected server. Click on the **Refresh** button to have an update occur. If the server resides on the local machine, the update will finish immediately. If the server is stored on a remote machine, the communication connection must be established first and then the update will take place.

Note:
The update will always be based on the contents of the file associated with the server subassembly. If the server subassembly is under construction, meaning the file is open, then only the status contained in the file can be accessed.

The second function allows you to assign data elements the server makes available to data elements contained in the client subassembly. A detailed description of this facility is available in the next section.

Server Instance Attributes

The server instance of the encapsulator component has three attributes, which are displayed on the attribute toolbar.

The annotation attribute is identical to the general annotation feature and is discussed in the general context. The same is true for the link facility.

The icon in the top left corner of the toolbar offers various settings for the window itself. The icon immediately next to it relates to the exchange of data between a client and a server instance.

The options available in regards to the window setting are

Caption: This allows you to enter a text which will appear as a caption on the window. The text is entered in the field entitled *Title*. While entering the caption it will appear on the little sample window in the bottom right section of this dialog box. Note that this field is evaluated through the parser. Therefore you may enter the name of another component.

Border: Selecting this option will let a simple border display at run time. The MINDMAP icon will appear at the left corner of the caption bar.

Dialog border: A dialog border will appear as a 3D beveled border. The MINDMAP icon will not appear on the caption bar.

System menu: This option toggle the standard system menu on/off. This menu offers the user various options at run time, including information regarding the MINDMAP application itself.

Maximize button: Selecting this option puts the maximize button on the right side of the caption bar. This allows the user to expand the application to full screen.

Minimize button: Selecting this option permits the user to minimize the MINDMAP application at run time.

Sizable: If you select this option, the user will be able to change the size of the window at run time.

Invisible: Selecting this option will keep server instance from displaying itself. It will remain functional, but cannot be seen. This option should not be selected if the server instance expects user interaction.

This dialog lets you modify the appearance of the servers window

This dialog box also offers the option of *disabling the client*, as long as the server instance is open. This is generally used to avoid conflicts in terms of entering data and thereby changing the status of the client instance.

Exchanging Data between Client and Server

Just as data can be passed between components on one page of a MindMap application or among different pages in an application, so data can be exchanged between components residing in different .MM files. The underlying principle is again based on the encapsulator component. The called upon subassembly exposes it data to the outside world via the encapsulator component. The calling subassembly can see the available data components and can opt to connect these to its own data components.

Thus data components can either be **local** to a subassembly or **global** in the sense that they are visible to other applications. But this is a restricted global view. The server is always in command in regards to when the data is supplied. The client can send a message to the server requesting data. This will not actually cause the data to be transferred, though. The server can now, on its own, use this message as an event and generate a message back to the client with the requested data attached. This is necessary to be able to eventually deal with asynchronous connections, in which an immediate response by the server can not always be guaranteed.

Begin by placing an input field in the client subassembly. Retain its default name (edt1).

Next place an input field in the server subassembly . Assign a link to it which, upon the start of the application assigns the parser statement Time to the input field. This will make the time display, once the application is put into run mode.
The next step is to expose the input field, containing the time in the server subassembly, to the client subassembly. In order to do this, access the attributes of the server instance of the encapsulator.

In the bottom half of the screen you will see the list of object assignments. Here is where any components that are visible to the outside world would be listed. At the moment, none have been exposed.

First, you must select a message onto which the data will be attached when sent back to the requesting client. Lets use the dummy message, This command does nothing…. Select it from the list at the top of the dialog box. As for direction, you can keep the bidirectional default setting for the moment. The next step is to attach the data to this message.

The name by which the input field is exposed to the outside world can be input in the **Description** field. In this case, it has been entered as ServerTime. Next we need to assign this (*arbitrary*) name to the actual component in the subassembly. Click on the button labeled **Object**. This will bring up the component list for this subassembly.

Select the input field and end by clicking on the **OK** button.

Note:
Data exchange is only possible between components which are capable of drag & drop, such as input fields, graphic components, data tables, list boxes, MCI, etc.

Note:
You can also assign by using the rubber band method.
The next step is to associate an event with this message. In other words, the server subassembly will wait for a data request via an event. It will then reply to the event with its own message and attach the data to it. In our example, we have attached the data to the dummy message:

This command does nothing…

We will use the incoming event:

Change my color to blue

as the trigger event requesting data. To do this, we will simply define a new link.

When the server instance of the encapsulator receives the event:

Change my color to blue

it will respond with the message:

This command does nothing…

with the data attached to this command. The data is the input field named:

edt1

This is all that has to be done in the server subassembly. Now, save the subassembly into its fileagain.

The next step is to make the necessary assignment in the client side. Open the client file and access the attributes of the client instance of the encapsulator.

The first step is to update the client instance connection to the server connection. Click the refresh button on the Data Exchange dialog box. This should result in the display of the following screen:
Now, the incoming data named ServerTime has to be assigned to a local component. Click on the **Assign** button. You will be presented with a list of all components in the client subassembly. Click on the input field named edt1.

Note:
You can also assign using the rubber band method.

Close this dialog box by clicking on the **OK** button. Save the client applicationand run it.
This has been a description of how data can be requested from a server. It is also possible to **send data to a server**. In such a case, it is not necessary to establish the call back facility.

Open a new file and place an input field. Then place a server instance of the encapsulator and define a dummy command. Assign the input field to the incoming global data component. Save the application into a file.

Open a new file representing the client side. Place an **Open** and a **Close** button and assign the appropriate links. Then place a **Send** button and an input field, the contents of which is to be transferred to the server.

Now place a link on the **Send** button so that the message you defined in the server will be sent to the server (*via the client instance of the encapsulator*).

The next step is to assign the input field to the global component, so that it can be transferred.

The internal component, named edt1, will be sent via the client instance of the encapsulator, as a component called Dollar.

Note:
In exchanging data, you must assure that the data types of the two components correspond to one another. MindMap does not perform an automatic type conversion.

Save this application in a fileand run it.

Note:
A client can never actually go out and get data from a server. It can only request the transfer of data from the server. The server must serve the data.

Default Events / Server Instance

Every server instance always registers one standard event. It is:

Server has Closed

This event can be used to send a message to attached clients, that the server has ceased operations.

Default Events / Client Instance

Every client instance registers two default events. These are:

Server has Closed

and

Server is Open.

User Defined Events

The user can define any number of events. In the context of the encapsulator component, they are referred to as commands. This is due to the fact that a command can be viewed as an event, as well as an action associated with a message, depending on the perspective.

Client/Server Component: Special Messages

If you have placed a **client instance** of the client/server component, two messages are automatically available. These are:

Open the Server

and

Close the Server

The first message will establish the physical and logical connection to the server instance and then open it. The second message will close the server instance and disconnect.

If you place a **server instance**, it will make the default message:

Close the Dialog

available. This will terminate the connection based on an action taken by the server subassembly.

Client/Server Component: Parser Extensions

The client/server component does not register any specific extensions to the parser.

Limitations

The current version of the encapsulator component does not support streaming data. Thus, it cannot be used to transfer live video, sound, or animation data.

If you are running MINDMAP as a centralized server for a group of concurrently accessing clients, a number of limitations may apply.

Since every client connects to a new instance of its corresponding server application, many server windows will appear on the server console, if the server application is defined to run *remotely*. It is obvious that the number of open server windows is limited by the available memory and system resources.
This restriction does not apply if a server application is *transferred* to the client computer and is executed there.

Be aware of the fact that other restrictions may apply due to limitations in the access to commonly used database resources, should your server application include a database component. (record locking, number of users, licenses, etc.)