

Effective Date: 09/20/2002

Supersedes: N/A

Revision History

<u>Revision</u>	<u>Date Issued</u>	<u>Description</u>	<u>Author</u>	<u>Approval</u>
1.0	09/20/2002	Original issue	S. Quayle	K .Wild

Notes:

.c.General;

This manual provides the information necessary to create, use, and maintain a graphical user interface using the Window Workstation program. It describes the process to reach these goals but does not supply many of the details, except where they are necessary. Refer to the *Window Workstation Reference Manual* if you have questions about a specific task or command. The release notes supplied with each Window Workstation installation contain information specific to a given release.

This manual contains the following sections.

Section Description

Using This Manual This section contains a brief overview of how the manual is laid out and what to look for in each section.

Product Configuration This section contains a step-by-step review of the initial configuration process.

Window Workstation Design This section contains a brief overview of the design of CRISPwindows and CRISPdraw.

Planning a Display System This section contains concepts that will reduce the time and effort required to implement and deploy a display system.

Detail Design This section contains detailed issues to consider before you create your displays. These issues include the following: using CRISP logic, displaying data, setting guidelines, display structure, security, performance, and printing.

Implementing Displays This section contains information that will help simplify creation of your display system.

Deploying Display Systems This section contains information necessary when deploying a display system regarding security requirements and configuration files.

System Administration This section contains the following System Administration information: configuring a display system, performance tuning, and security administration.

General (cont)

Section Description

Run-Mode Usage

This section contains the following Run-Mode information: starting Window Workstation, logging in, WWS controls, navigating between displays, exiting and/or logging in, and using Window Workstation without a mouse.

Problems This section contains a listing of common problems encountered in Window Workstation and provides how to correct the problems.

Glossary This section contains definitions of terminology used in this manual.

.c.General;

Each section of this manual begins with a summary of the essential ideas within the section. Please read the summarized material to quickly get the essential ideas of this manual.

We designed this manual, in conjunction with the *Window Workstation Reference Manual*, for user interface designers, maintainers, and users. This manual presents and describes the tasks required to create, administer, and maintain a user interface; whereas, the reference manual describes the detailed technical information for a specific task.

We organized this manual primarily around the steps and tasks necessary to develop a successful user interface. These consist of planning, detail designing, implementing, deploying, and maintaining. The other sections pertain to administering and using Window Workstation.

Your role in creating, administering, maintaining, or using the user interface system determines which sections of this manual will be most valuable.

- Designers should read the planning, design, and implementation sections first.
- System administrators should read the configuration and administration sections first.
- Operators should read the run-mode section first.
- Maintainers should read the design and implementation sections of the manual first.

Despite the linear ordering of the subjects in this manual, developing a real user interface seldom progresses as smoothly as described here. Typically, you may perform several steps, find yourself in the wrong place, and must backtrack a couple steps. We hope presenting issues influencing later steps reduces the amount of backtracking (and cost) necessary to create and deploy a user interface.

At each step, we present issues we have addressed in our own user interfaces. Some of these issues reflect problems that should concern you at the present step of design and construction. Other issues reflect possible future problems you can solve most easily in the present step. Some issues may not relate to your system. In any case, we hope these issues cause you to consider your design and improve it in some fashion.

We have inserted throughout the text some pointers to other sections using the note 'Refer to'. These other sections contain information that have similar concepts, issues, or precede or succeed a step. They can help you follow a concept through the various sections and get a complete understanding of the issues.

Notes:

.c.General;

The Window Workstation product must be configured after installation. There are two major steps: executing the CRISP_CONFIG_WWS command procedure, and then configuring various data files. The data files have no step-by-step procedure, but are detailed in the remainder of this manual.

Central Administration; The CRISP Central Administration product has its own installation/configuration guide. If Central Administration is used, the guidance in that document must be used.

Enhanced Security; The CRISP security facility, which is used by the Window Workstation product, can be enhanced by installation of a CRISP-WWSPLUS license key. The design of displays is the same; however, the Enhanced Security option allows the Window Workstation product to produce audit trail records that are intended to comply with 21 CFR Part 11 requirements.

To use Enhanced Security, all CRISP systems must be running the CRISP Log Server, and capturing all records to disk files. Consult the CRISP/32 configuration manual for more on the CRISP Log Server.

CRISP_CONFIG_WWS; The Window Workstation product has a configuration procedure, which covers various defaults and options. The questions and possible answers are covered here, step by step. To begin the configuration process, execute the following command.

```
$ @CRISP$:CRISP_CONFIG_WWS <Ret>
```

The command procedure will ask the following questions. The default values, if not previously configured, depend on whether or not the CRISP-WWSPLUS license is installed. Pressing “<Ret>” at each question selects the default presented in square brackets.

```
Password minimum length, characters . . . . [8]:
```

Passwords can be required to be a minimum length. A value of zero allows passwords of any length. The largest possible password is less than 256 characters in length.

```
Password lifetime default, days (0 = none). . . . [90]:
```

Passwords can be set to expire on a regular basis. When the user attempts to log in, and his password has expired, he is forced to select a new password. A value of zero means that passwords will never expire.

```
Password failures before lockout (0 = none) . . . . [5]:
```

After the specified number of consecutive failed passwords, the user will be prevented from logging in, even with the correct password. A failure email can also be sent to a system administrator. A value of zero disables this feature.

```
Require mixed passwords . . . . . [Y]:
```

Passwords are not case sensitive (“a” is the same as “A”), and can have letters A-Z, digits 0-9, and the underscore (“_”) character. A “mixed” password is one that contains at least one letter and at least one digit, subject to any minimum length.

```
Require employee ID's to be unique . . . . [Y]:
```

Each record in the CRISP security file carries an employee ID value. If this option is enabled, it will not be possible to add a new record which matches another record's employee ID.

```
Minimum WWS autologout time, minutes . . . . [10]:
```

If non-zero, and if there is no mouse or keyboard activity, any user logged in to the Window Workstation will be

automatically logged out. The user "DEFAULT" will be logged in instead. A value of zero disables this action.

CIA_MAINT_WINDOW autologout time, minutes [5]:

The CIA_MAINT_WINDOW program is used to create and modify all usernames in the CRISP security domain. Because of the power of this program, a non-zero value is recommended. After the specified period of inactivity, the program will exit. A value of zero disables this action.

Email address to notify for security events [SYSTEM]:

If the CRISP-WWSPLUS license is not installed, the above question is not asked. When a user becomes locked out due to too many failed password attempts, an email is sent to the specified user, which could be a distribution list. A value of "-" disables this feature.

WWS bias value, bits [0]:

To create multiple security sub-domains, the 32 bits of Window Workstation privileges can be broken into groups. The privilege value from the CRISP security file is shifted right by the bias value before comparing with the required bitmask in the WWS display file.

Central administration node [XYZ]:

Enter the DECnet node name of the central administration node. A value of "-" disables central administration.

Add a proxy for the central administration node [Y]:

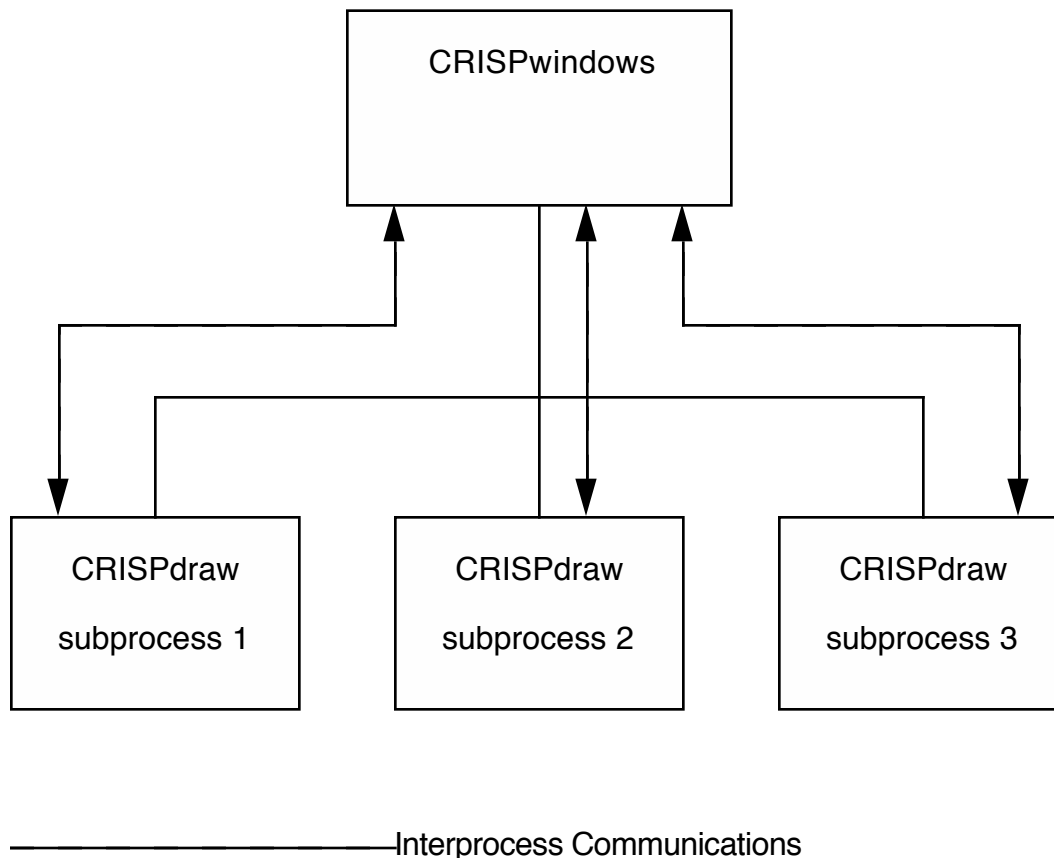
This allows the central administration node to update the local copy of the CRISP security files on a periodic basis. This question is asked only if the central node was not set as "-" previously, and if the node is not the Central Administration node.

Suppress logging of WSDIR changes [Y]:

When the Window Workstation changes screens, CRISP database WSDIR is modified. If operator action causes the screen to change, it can be logged. By answering yes, the change is not logged.

.c1.General;

The Window Workstation product includes two executable programs, CRISPwindows and CRISPdraw. The CRISPdraw program runs as a subprocess of the CRISPwindows program. Each CRISPwindows process can have one or more CRISPdraw subprocess as shown in Figure 1. Inter-process communications methods handle communications between the two programs. CRISPdraw uses the standard .i.X Window;.i.X Window library;X Window library for graphics and CRISPwindows uses .i.Motif;Motif. This means the programs may execute on one computer and use another workstation or an X Window terminal as their display device. Only one .i.CRISPwindows:limits;CRISPwindows program may execute on a given display node at any given time.



.c1.Figure 1. Window Workstation Architecture;

Notes:

.c1.General;

Good planning reduces the amount of effort and time required to implement and deploy a system, while increasing user satisfaction with it. A study [Niels91] shows that several methods or activities positively impact user interface development. Several of these activities must be done before the user interface is developed.

The activity having the most benefit is analyzing the user's current task. This means knowing who constitutes the user and the tasks they perform.

A second beneficial activity is iterative design. The first step in an .i.iterative design; is creating a prototype. The prototype will aid your planning, help solidify concepts, and discover unresolved issues.

This section describes issues you should address while creating the prototype. Addressing these issues at an early stage in the development will help you make correct decisions earlier and should reduce the amount of rework necessary to perfect your user interface. Each hour spent planning your display system can reduce your work by many hours later in the process.

At this stage, you may not be too familiar with the capabilities and limitations of Window Workstation. Resist the urge to create displays at this point. Instead, skim the remainder of this manual and the *Window Workstation Reference Manual* to get a feel for the capabilities and limitations. You may also want to review the example displays to see Window Workstation in action. Remember, these displays were built for example purposes, not to run a process. There is much that needs to be changed for them to be effective process control displays.

.c1.Know Your Users;

The most important person you should be concerned about during the design of your user interface is the user. The user is the one who must use your design every day and your decisions can either make their job easier or needlessly complicate it. Some issues you should consider at this stage are identifying your users, your users' characteristics, the tasks your users will perform with your system, and any limitations your users may have.

.c1. Identifying Users;

It may be difficult to determine who are the users of your .i.user:display system;. An obvious group of users is the process operators. Other users may include process engineering, maintenance, supervisors, and plant management. Each of these users will place different demands on your display system and will have different expectations. Identify each group of potential users of your system.

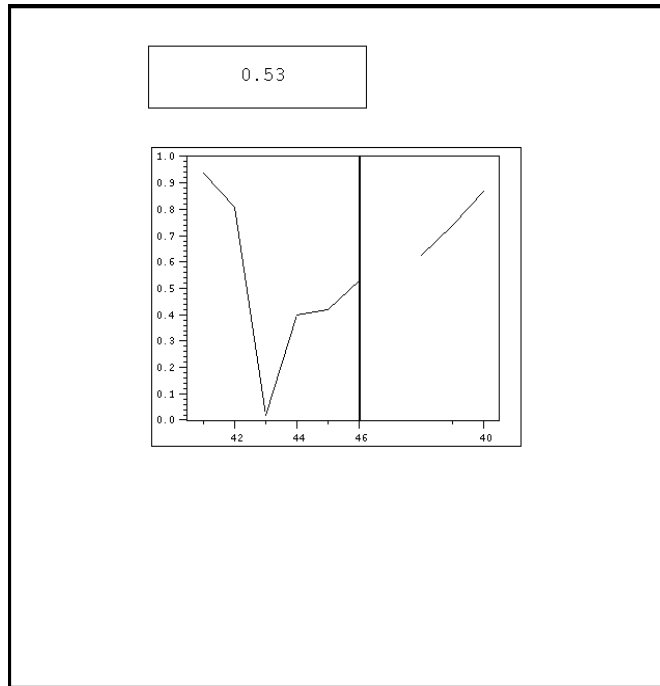
.c.User Characteristics;

What are the major .i.user characteristics;characteristics of your users with which you should be concerned? This may include their education level, the amount of information they can comfortably keep in their heads at once, their operating environment, and other such characteristics. Do this for each group identified in the previous section (refer to Identifying Users).

.c.Tasks Users Must Perform; When a user tries to do a .i.user:tasks;task or reach a goal, there are several implicit steps that must be performed before the goal is realized. The first step is to gather data relative to the system and the user's goal. The second step is to evaluate the data to determine the state of the process and where it is heading relative to the goal. The third step is to formulate a plan of action based on the information obtained in step two. The last step is to carry out the plan of action.

One of your goals as the designer of the user interface is to make it easy for the user to determine the state of the system under control, to evaluate the state of the process relative to a given goal and to formulate a plan of action to reach a given goal. Some examples of how you can make it easy for the users to do these steps are as follows.

- Present all of the information a user requires to reach a goal on one display. Most people can keep only five to seven pieces of data in their heads at once. Therefore, it is counterproductive to put all of the data a user will need for several tasks on one display and expect the user to sort out the data for a particular task. If a task requires more than seven pieces of data during a particular task, consider combining two or more of these into a usable composite value.
- Another way to limit .i.data overload:limiting;data overload is to emphasize the most important data and de-emphasize the less important data. This guides the user to the most important information by highlighting it while quickly giving the user access to the less important, supporting data. Refer to Coding Standards.
- .i.data presentation;Data presentation can also impact the users performance. Digital meters are relatively poor in conveying rate information while line graphs or meters are much better see (Figure 2). Refer to the Data Formatters section.



.c1.Figure 2. Digit Graphs vs Line Graphs;

.c. User Limitations; User *.i.user:limitations;* should be considered when designing a display system. Some of these limitations are covered in this section. You may be aware of others in your operating environment.

.c. Visual Limitations; A common visual limitation is poor eyesight. If a user has uncorrected vision, increase the *.i.font size;*font size of the text on your displays to compensate. Also increase the font size if the display must be viewed from a distance.

Another common visual limitation is color blindness. The most common form of color blindness involves being unable to distinguish between the colors red and green. Less common is an inability to distinguish blue from green.

.c. Operating Environment; The user's *.i.operating environment factors;**i.user:operating environment;* should also influence the design of your user interface system. It is common for users working within a *.i.hostile environment;* to wear protective clothing. This reduces mobility and sensitivity, and increases the amount of space required for pushbuttons, for example.

The environment may also require a protective eye wear that may reduce display visibility. Increase the size of the text font to compensate.

.c. Prototyping Displays; An effective display system is difficult to create without first *.i.prototyping displays:goals;*prototyping it. The *.i.Prototyping Displays:goals;*goals for a *.i.prototype;* are as follows.

- Determine whether the system is complete.
- Estimate how easy the system will be to use.
- Determine if there are any dead ends in your display network.
- Define additional issues in the display system.

Sketch your ideas on paper with as much detail as you can provide. Mentally walk through all of the tasks the system must perform. Try multiple ways of doing a particular task, then select the one that is easiest from the user's point of view. Show your sketches to the user community and other interested parties to get feedback. They will provide plenty of opportunities to improve the design. You may also want to use CRISPdraw to prototype your displays (refer to the CRISPdraw Prototyping section).

The display system design may require several iterations because each iteration will discover new issues to resolve. Strive to get the most from each iteration and make changes where they are required. The purpose of the prototype is to experiment with different designs and select the best one.

.c.Pencil And Paper Prototypes; Pencil .i.prototyping displays:sketches;sketches on paper or notecards make an effective first .i.prototype;. There are a number of advantages to this method. The prototype is cheap and easy to change, it is easily understood by people who are not computer literate, and the technology is less of an issue.

Your sketches will be very rough in the beginning, perhaps consisting of only a name and description on each page. Add more detail as you progress. The primary focus at this stage is to define the .i.display system:structure; of your display system. Use role playing to put yourself in the position of an operator using your system. This will indicate to you the effectiveness of the display system to control a process or gather information about the process.

.c.CRISPdraw Prototyping; You can also create and refine prototypes .i.prototyping displays:using CRISPdraw;using CRISPdraw. The advantage to this is you see the appearance of your displays as they will appear to the end user. You can also obtain a rough estimate of the display speed. The major disadvantage is that you can spend too much effort refining minor aspects of your display system.

As with any prototype, strive to keep the system as simple as possible. Instead of using CRISP databases and variables, which might not exist at this stage of a project, use file data sources and function data sources to simulate the data. You should also keep the number of databases to a minimum.

Also keep all of your graphics as simple as possible. Consider the following techniques.

- Use hardware text instead of vector text.
- Keep pushbuttons simple.
- Use text labels instead of fancy graphical objects.

Design your displays so they are compatible with the .i.prototyping displays:target compatibility;target system. This is especially true with respect to the .i.monitor size;. Common monitor sizes are 640 by 480, 1024 by 768, 1024 by 864, 1152 by 900, and 1280 by 1024. The number of colors can also cause some problems. For .i.Printing:optimal display size;printing using a .i.Digital LJ250; printer, a display size of 900 by 720 is optimal. Refer to the .i.Printing; section.

.c.Get User Feedback; Once you have sketched the basic system, show it to the users to get their .i.user feedback;feedback on the system. This will help you uncover problems and improve your system. It will also uncover additional issues you may not have considered.

.c.Try To Attempt To Do The Tasks That Need To Be Done

Use the prototypes to try to perform the tasks you have identified for your display system. You may locate awkward display sequences and individual displays that require revising.

Modify the prototype until you are satisfied all tasks can be done easily and efficiently.

.c.Look For Flow Between Displays And Objects, Dead Ends, and Traps

Look for several problems that can be corrected easily at this stage. First, eliminate .i.Dead-end displays;dead-end displays (i.e., displays a user can enter, but not exit). It is easy to create a display that the user can enter, but the security system will not allow the user to exit. Detecting this situation may be difficult.

.c.Diagramming; Diagramming helps you locate dead ends and provide .i.display navigation;navigation for users. Draw a map of your displays to locate dead ends and provide a .i.user:navigation; map for users. Represent each display by a circle or a box and each **GOTO** run-mode action by an arrow from one display to another, as shown in Figure 3. Any circle with arrows pointing inward and none pointing outward will be a dead end. You may also use a coding scheme, such as coloring the arrows or adding symbols, to help you detect a dead-end situation.

.c.Try Extraordinary Tasks; Avoid the trap of .i.clumsy automation;; common tasks are easy to perform, but emergency or critical operations are more difficult. This occurs when insufficient attention has been given to tasks required during .i.Emergency operations;emergency, .i.Critical operations;critical, or .i.Degraded operations;degraded operations. Think through the tasks required during these times and make sure operators can safely and easily perform these tasks.

.c.Organizing Displays;When planning your project, you should be concerned with how your .i.Display organization;displays relate to each other and to the tasks your users perform. The two primary ways to organize a set of displays are as a .i.hierarchy; or as a .i.network;.

The hierarchy method is best for small numbers of displays because it provides an implicit means of navigating through the display system. Its primary disadvantage is that it can require several display switches to get to the next display in a task sequence.

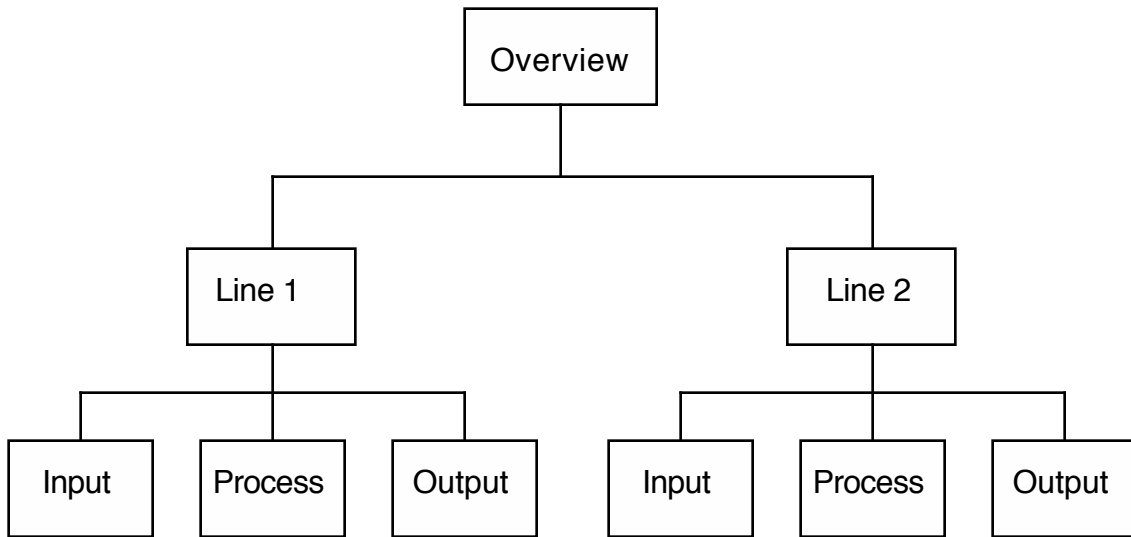
The network method is best for larger display systems because it can ease the transition between displays for a given task, but it is easier for the user to get lost within the display system if navigation aids are not provided.

.c.Organizing Methods For Multiple Display

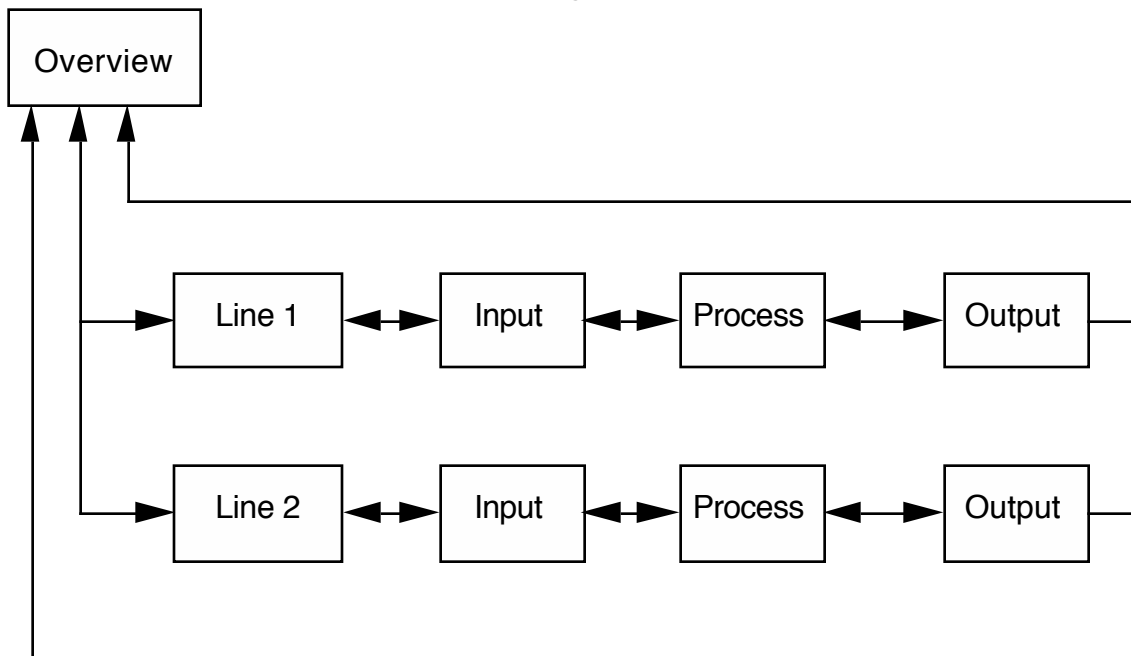
Organizing Methods For Multiple Display

If the display system has more than just a few displays, provide a way for users to .i.display navigation;navigate through the display system. The two primary methods of organizing displays into a display system are the .i.display organization:hierarchical;hierarchical and the .i.display organization:network;network organizations. Each of these methods is suited to particular application requirements. The .i.hierarchical organization; is better for small systems of displays. The network is better for larger collections of displays. Figure 3 shows the two primary organizations.

Hierarchy Organization

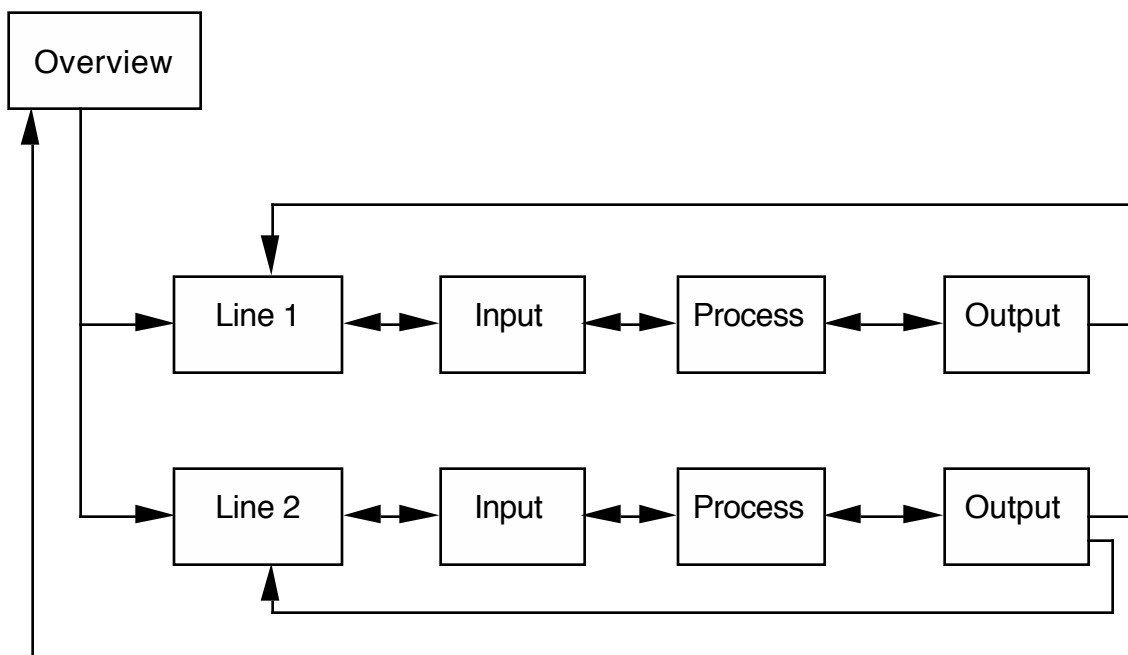


Network Organization



.c.Figure 3. Hierarchy vs Network Display Organization

- .c.Hierarchy;** The .i.hierarchical organization; is better for small systems of displays, that is, systems with around ten or fewer displays. The advantage of the hierarchical organization is that it is easy to comprehend and the user can easily go back to a known point by simply going up in the hierarchy. The hierarchy also makes it easy to locate a particular display by simply traversing the system, although this can be very time consuming. The disadvantage of the hierarchical organization is it can be very difficult to switch to a particular display when there are time pressures. If the information a user requires to complete a particular task is spread among two or more displays, this method can become extremely awkward and can significantly increase the chances for errors.
- .c.Network;** The .i.network organization; is a better means of organizing large numbers of displays, but users can get lost in it easier than in a hierarchical organization. It is also much easier to design systems in which there are paths that trap a user, as shown in Figure 4. One of the best means of avoiding these problems is to diagram your network of displays as you create them. This will instantly show you if you have a path of .i.entrapment; and it will also serve as a diagram for the user.



.c.Figure 4. Dead-End Display Paths

.c.Display Layout And Design; Most display systems will naturally gravitate to the network model of .i.network organization;, so it will be the model that is used in this description.

Designing the display network involves analyzing the tasks the operators will perform, determining the steps required within each task and designing a display system to assist the operator when doing each of these steps.

Each individual display within an organization should be designed so it is related in some way to .i.neighboring displays;.

Each individual display should be designed so the user can easily grasp the information contained in it. This means the designer must be cognizant of what data the user needs for any given task and how that data can be most effectively presented. The user must integrate all sources of data into a .i.mental model; of the process and devise a plan of action to reach a particular goal.

.c.Task Analysis And Process Flow; The first major problem for an interface designer is listing and analyzing all of the .i.analyzing tasks; that operators of the system will be required to perform with the system. This may involve interviewing users of the current system if a system is being upgraded or replaced, interviewing process control engineers if a new process is being built, or it may involve reading the specifications and requirements for a new process if the process control engineers are unavailable.

Once you have all of the tasks required to operate a process, analyze each of the tasks to determine the ordering of the steps within a given task and the information required to complete each step. You can then use this information to estimate the number of displays you will need, what data each display will contain, and how displays relate to each other.

In addition to the usual tasks, you should also consider .i.emergency operations; and unusual tasks. Make sure these are easily done with your display system.

Constantly evaluate your displays at this stage by asking yourself if the user will have all of the information necessary to make a good decision. Is the information distributed across two or more displays? If so, why? Can it be consolidated into a single display? Can any of the data requirements for a step be simplified? Can the displays for two or more steps be combined into a single display, thereby decreasing the number of displays to implement?

Keep detailed notes of what information each display contains and what step of a task a display represents. Detailed notes will benefit you later in the design process and benefit those who must maintain your design in the future.

.c.Overview vs. Detailed Displays; Displays can be broadly categorized into .i.overview; or .i.detailed displays;. Each of these is useful in a particular context. The overview display is useful for giving the operator an overall view of the process while the detailed display should give the user a small view of a part of the process.

(Continued on next page.)

Overview vs. Detailed Displays; (cont) The overview display will necessarily contain much less information about any given aspect of a process. Only the most important information about a process should be included in the display. The operator can go to the detailed displays to get the more specific information. The overview displays should have pushbuttons on them that will switch to the detailed displays when pushed.

The detailed displays should contain only the information necessary to complete one or more steps in a task. The amount of information on any given display should be limited to prevent the operator from becoming overloaded with data. If too much information is required you may want to check the step to see if it can be broken into two or more steps with less information required for each of the new steps.

.c.Display Navigation;

With any display system composed of more than just a few displays, provide some means of display navigation for the user, especially when a network organization is used. This can range from presenting a map showing where the user currently is in the network to providing meaningful titles that incorporate the location in some coded form.

The overview displays can be used to provide a means of orienting the operators and providing a means of selecting a particular display.

.c.Navigation Aids; A number of techniques are available to assist the user in navigating through a display system. These include providing a map of the display system, providing contextual clues within a display, help screens, and pushbuttons that transport a user back to a particular point. Each of these different methods has advantages and disadvantages and their use should be decided using the context within which they will be used.

The graphical map technique is good for positioning a user within the context of the entire display system. It should show the user where he is relative to neighboring displays. A problem with this technique is determining the number of neighboring displays to show and deciding the methods to indicate the buttons or function key necessary to go to a neighboring display.

Providing contextual clues means providing some overlap with neighboring displays so a user can see exactly where he is relative to the overall task or step. This usually means providing some graphical objects from neighboring displays around the edges of the current display. The disadvantages to this system are that it may be difficult to define unique contextual clues in some situations.

Help screens can also be provided that show a user's location in the display system. This display may be as detailed as necessary, since it can be a single display. The disadvantage to this method is that a number of help displays must be created.

(Continued on next page.)

Navigation Aids (cont) Pushbutton labels may also be provided that indicate where a user is in the display system.

You can also use background color to designate to which line a particular display belongs when an operator controls multiple lines. This makes it easy for the operator to determine which line is being controlled by the display.

The most effective system will combine several of these elements in a single display.

.c.Subdrawings;

.i.Subdrawings; are portions of displays that can be used in multiple displays. These can be used to reduce the amount of work you must do to create a display system.

You can import subdrawings into displays in two ways, referenced and included. If you want the same subdrawing to be used on a number of displays and you may change the subdrawing in the future, then import it as a referenced subdrawing. When you change the subdrawing, all of the displays that use the subdrawing will be automatically updated.

The included subdrawing is used when you have a subdrawing that is substantially the same on each display, but has minor differences between the displays. Any changes made to the subdrawing after you include it will not be reflected on other displays.

With either of these methods, the .i.color dynamics; and .i.data sources; for data formatters within the subdrawing will be lost when it is imported into the display. If you wish to preserve the color dynamics and data sources, merge the subdrawing into the display. All of the entities within the subdrawing will then become entities in the display.

Refer to the Drawing within the Complex Editing subsection of the General Editing section in the *Window Workstation Reference Manual* for further information about subdrawings. Refer to Merging View within the Using the Commands Menu subsection of the General Editing section in the *Window Workstation Reference Manual* for further information about merging views.

.c.Set Standards;

At this stage of constructing the display system, you should be aware of existing standards and decide whether following the standards will be advantageous. There are both advantages and disadvantages to using standards.

In general, .i.standards; make the system easier to create, learn, and use. They make displays easier to create because they reduce the need for experimentation. Standards can reduce the learning requirements for a display system, since the new system has some resemblance to a previous system. Standards also increase the consistency for users, thus reducing the amount of experimentation users must perform to become proficient.

(Continued on next page.)

Set Standards (cont)

The disadvantages of standards are that you may not have access to pertinent standards, standards may be difficult to use, and existing standards may be inappropriate for a particular purpose. The organizations listed in Appendix A either have existing standards covering user interfaces or they are actively working in the area. You can get the latest information concerning their work by contacting them at the addresses listed.

Two papers, [TETZ91] and [THOV91], explore the use of standards to create user interfaces and conclude standards are difficult to apply, even though the designers are willing and motivated to use them. Problems mentioned with existing standards include difficulties locating relevant information, the volume of information contained in the standards, and a lack of good examples.

Standards may be inappropriate for several reasons. For instance, the standard may address outdated technology. A standard may cover situations that will not occur in your environment. Finally, the standard may be addressing the wrong audience; a standard for office automation systems may be inappropriate for a factory floor system.

.c.Coding and Usage Standards; Coding and .i. standards:usage; .i. Standards:coding; standards are conventions defined for the various types of symbols present in the display system.

The purpose of coding standards is to reduce inconsistencies and make it easier for the user to ascertain the meaning of a given display. The purpose of usage standards is to make the display system usage much more consistent from a user's perspective.

.c. Standards Organizations; .i. Standards organizations; and industry have adopted some standards. One standard, ANSI/ISA-S5.1-1984, describes computer terminal .i. Standards:symbols; symbols for process control components and should be consulted for process control systems. Either the Instrument Society of America (ISA) or the American National Standards Institute (ANSI) can provide information about the current version of this standard (refer to Appendix A for the addresses of these organizations).

Other organizations especially involved with computer human interaction are the Human Factors Society (HFS) and the Association of Computing Machinery (ACM), which has a Special Interest Group on Computer Human Interaction (SIGCHI) devoted to this field. The SIGCHI has a newsletter with a column devoted standards.

Other organizations working in this area include the International Organization for Standards (ISO), the Canadian Standards Association (CSA), the European Computer Manufacturers Association (ECMA), the Institute for Electrical and Electronics Engineers (IEEE), the International Electrotechnical Commission (IEC), and the International Federation of Information Processing (IFIP).

.c. **Coding Methods;** There are three primary means of .i.encoding information; on a display. These are shape, color, and size.

The shape of a graphical object gives a direct clue as to its meaning. An excellent example of this is the roadway signs; all drivers know the shape of a stop sign. Another example is the graphical pushbuttons that are designed to look like its mechanical counterpart. This gives the user an intuitive feel for the operation of that pushbutton. Modern windows based systems are also examples of using shapes (icons) to indicate functionality.

Color is also used to indicate meanings. In most Western cultures, the color red generally means danger, green generally means ok, and yellow generally means caution. Corporations are identified with colors through their use in their trademarks and logos. Examples include IBM (Big Blue), McDonalds (Yellow Arches), Pepsi (red, white, and blue), and Coca-Cola (red and white).

Size can also be used to encode information, especially relative quantities.

.c. **Usage Standards;** .i.Standards:usage;Standardize the 'feel' of a display system, so a user knows what must be done on each screen. We developed the following guidelines to assist us when creating displays. These guidelines apply equally well when designing and implementing the display system.

Put similar objects in similar places on each display so the user can rapidly locate them. For instance, put the Next and Previous button in the same place in each display of a series. A user can then quickly traverse the displays without moving the cursor.

Define guidelines so a given symbol, function key, or other element has the same meaning on all displays. When a user then views a display for the first time, the symbols on the display will make sense.

Try to design your system so changes to strings occur at the end of the string. The user can change the value more easily when changes occur at the end. Then all a user must do is delete the ending characters and type the new characters.

If the same information or symbol appears on each display, it should appear and act the same on each display. This increases the consistency and predictability of the display. You can merge displays or use subdrawings to reduce your effort to implement this guideline.

You can use background elements to indicate the action of a display symbol. The background shape or color can be used to encode the meaning of the foreground object. Give the background object the same run-mode action as the foreground object, especially when the foreground object consists mainly of lines on top of empty space.

.c. Displaying Data;

Window Workstation provides a number of sources for data including CRISP real-time databases, CRISP trends, data files, and internal functions.

Two .i.important attributes; of individual pieces of .i.data; are the type of the variable and its shape. The type of the variable refers to whether the variable represents a floating point number, an integer, or a character string. The shape of the variable refers to whether the variable is a scalar, an array (or vector), or a matrix.

Data formatters generally work with only limited types of variables. For example, a digits graph only works with Numeric types and a text graph only works with text Strings.

Data formatters are also designed to work with specific variable shapes. For example, a line graph works with both scalar and vector quantities, but a trend graph only works with scalar quantities.

.c. CRISP Data Sources;

Displays access CRISP real-time variables through a CRISPdraw real-time data source whose name contains the node and database name of the CRISP real-time database containing the variable. You can access any .i.CRISP real-time database; on the network or local machine.

Use a symbolic name for the node name and enter the symbolic name along with the real name in the .i.CRISPWIN_WORF_ALIAS.DAT; file (refer to .i.CRISPWIN_WORF_ALIAS; file). This lets you easily change the database location should it be moved from one machine to another.

The .i.CRISP real-time database; supports accessing scalar, array, or individual array elements.

Accessing .i.CRISP trend data; is similar to accessing real-time data. The node and database names are used in the trend data source name, and variable names are added within the data source. When accessing permanent trend data sources you must define the number of samples per point and the number of seconds per sample exactly the same as the permanent trend; otherwise, a temporary trend will be created.

.i.Trend variables; are maintained as arrays of CRISP floats by the trending process. The number of elements in the array depends on the parameters used to create the trend data source.

The .i.Trends:advantage; of using trend data sources is that it provides a recent history of a CRISP variable. This enables a graph to display some history when a user switches to a new display.

The .i.Trends:disadvantage;s of trends include resource limitations and changes of trend data. Each trend data source requires a substantial amount of data to be transferred from the trend region to CRISPdraw. This increases the CPU and memory requirements of the workstation used to process the data. Temporary trends can also be deleted when all of the trend process slots get filled. Refer to

the CRISP system documentation for additional details.

- .c.Other Data Sources;** Several other data sources are available including data files, internal functions and constants. .i.Data files; contain values in either binary or ASCII format. You must provide the structure of the file when creating your data source in CRISPdraw.
- .i.Function data; sources compute data values using a combination of internal functions and arithmetic operations. Data may be obtained from other data sources and combined within these function data sources.
- The final type of data source is the .i.constant;; it is used to provide constant values for other data sources and data formatters.
- .c.Data Formatters;** A wide variety of .i.data formatters; are available to display data contained in data sources. These range from the digits graph, for displaying numbers, to Statistical Process Control charts. Each graph has its own requirements for the .i.data:type; and .i.data:shape; of data. Refer to the Graph Types section in the *Window Workstation Reference Manual* for more information about individual data formatters.
- .c.Data Formatter Usage;** There are several principles to guide data formatter selection and usage. A brief description of these principles follows; the Detail Design section of this manual contains a more detailed description.
- Use a .i.digits graph; when data must be displayed to a high precision. Use a graphical data formatter, such as a line graph or bar graph, when the rate of change of data is important.
- Use a .i.line graph;; or one of its variants when the operator needs a historical context to the current data. A line graph is preferable to a .i.strip chart; because the line chart requires significantly less CPU usage and graphics operation.
- Design your data formatters to use an entire array. If you require a .i.array element:subset; of the .i.array element;s, you must use CRISP logic to extract the element subset to a new array unless the subset begins with the first element of the array.
- .c.Color Dynamics;** You can change the color of individual graphical elements in CRISPdraw using .i.color dynamics;. Color dynamics use a .i.color threshold table; to associate the value of a data source variable to the color of an object.
- .c.Drawing Dynamics;** .i.Drawing dynamics; are similar to .i.color dynamics;; except they associate a number of subdrawings with the value of a .i.data source variable;. This lets you change the appearance of an object based on the value of a data source variable.

.c. Planning Checklist

- Users are identified
 - User characteristics
 - User tasks
 - User limitations
 - User environment
- Display system has been prototyped
- Display system has been critiqued by end users
- Process control operation has been simulated
- Display organization has been diagrammed
- Display system navigation aids are provided
- Standards have been addressed
- Data formatters address the tasks required

.c.General;

Once your prototype stabilizes, consider additional issues before you create displays. You may need to use CRISP logic to support some display functionality. Consider the source of the displayed data and organize your data sources to minimize the typing required and simplify maintenance. Formulate guidelines concerning color usage, symbols, logical names, and other such items to give your display system consistency. Organize the structure of each display so it conveys a maximum amount of information with a minimum amount of effort by the user. Consider security issues and determine how these affect your displays. Your performance requirements will impact your display system. Your requirements for display hard copies can also affect the design of your displays.

.c.Using CRISP Logic;

Certain display operations, such as accessing Historian data, require you to use .i.CRISP logic; statements. Other display operations can be considerably simplified by using CRISP logic constructs, such as displaying alarms.

.c.Displaying Data;

CRISPdraw can display data from a variety of sources in a number of different formats. The usual source of data is a .i.CRISP real-time database; or a .i.CRISP trend; process. The .i.CRISP HISTORIAN; is not supported directly, but it can be accessed by using the SCAN_HIST call in CRISP (refer to HISTORIAN Access).

Historian data can also be displayed by using the Historian Viewer layered product.

.c.CRISP Data Sources;

You should be aware of several items when using CRISP data sources. This section describes some limitations and conditions when using CRISPdraw to display CRISP data.

- .c.Real-Time;** CRISPdraw supports all .i.CRISP data types; except counters and timers. If you want to display these data types, you must use a long variable to store the set or countdown value for either of these types in your CRISP logic.

If you wish to display a range of elements of an array, you must define an array in your .i.CRISP database; to store the elements and use .i.CRISP logic; to move the elements from their original arrays to the display arrays since CRISPdraw is incapable of accessing ranges of elements of an array. It can either display an entire array or an individual element of an array.

You may use any CRISPdraw data source variable as the .i.CRISP array element:index; for a .i.CRISP array element; by typing the name of the data source variable in the array index. The .i.function data source; is especially useful since you can have it calculate an index. This reduces the need for special CRISP logic to support the user interface.

- .c.**Trends**; Despite their close names, .i.trend data sources; should not be displayed in a trend data formatter. The .i.trend data formatter; was designed to display scalar quantities and not arrays. The trend data source returns only arrays, thus the two are incompatible.

When you specify a trend data source, it transfers the entire contents of the trend buffer to CRISPdraw. CRISPdraw then selects the last N points, corresponding to your specification and displays them. Refer to Trend Chart in the Graph Types section of the *Window Workstation Reference Manual* for more details.

- .c.**Other Data Sources**; CRISPdraw features other data sources including the function, constant, memory, file and process data sources. Of these the function and constant data sources are most useful for most CRISP applications.

- .c.**Function Data Sources**; The .i.function data source; is useful for controlling display or drawing dynamics by allowing you to calculate quantities without resorting to writing CRISP logic. A number of useful functions, such as sine, cosine, tangent, and square root, can operate on values from other data sources.

Some uses for the function data source include calculating .i.array indices; and generating values for color or dynamic threshold tables.

A limitation of the function data source is that only two values can be imported into a given function data source variable. This means that if you have a calculation involving three or more data source variables, you must either use two or more function data sources in stages, or perform the calculation using CRISP logic. Refer to Function Data Source Usage subsection in the CRISPdraw Usage section of the *Window Workstation Reference Manual*.

- .c.**Constant Data Sources**; The .i.constant data source; is useful when you have a constant numerical or string value that will not change during the execution of the program, such as parameters, labels, or titles. The constant data source can be especially useful with the .i.moving drawing data formatter; if you want to change only one attribute of the moving drawing, but leave the other two alone. For instance, say you wish to scale the drawing, but not rotate or displace it. You can bind the rotation and displacement attributes to one or more constant data source variables and bind the scale attribute to another data source. Your drawing will then be scaled, but will not be rotated or moved.

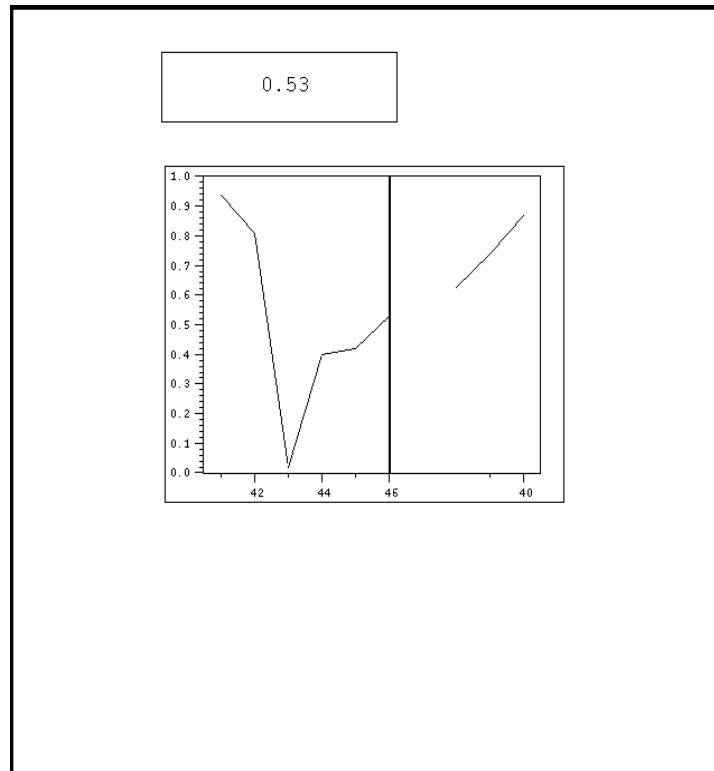
- .c.**File, Process, and Memory**; CRISPdraw offers other methods of getting data, but they are not generally used within typical CRISP applications. Of these other methods, reading information from a file is the most useful, but you must have some means of generating the file. It can be either a binary or an ASCII file, but when CRISPdraw reads it in, it will be read sequentially. Refer to the *Window Workstation Reference Manual* for more information.

.c.Data Formatters;

The most useful .i.data formatters; are the .i.line graph;, the .i.digits graph;, and the .i.text graph;.

The .i.line graph; is most useful for displaying data within a historical context, but at a fairly low precision. These uses include trending, SPC plots, and rate of change plots. Thus the line graph is most useful for showing where a variable has been and for giving the operator an idea of where the variable is heading. The line graph can have only one scale associated with it, therefore all of the variables that appear on the graph should be scaled to the same range.

The .i.digits graph; should be used to show values to a high precision, but where the historical context of the variable is not as important. The digits graph can be combined with the line graph to provide an operator with a historical context as well as a high precision current value as shown in Figure 5.



.c.Figure 5. Digit and Line Graph Combination

The text data formatter is useful for displaying strings.

Other graphs can be used within a display, but the above three are most commonly used. There are data formatters that are derived from the .i.line graph;, such as the .i.point-line graph;, .i.filled line graph;, or the .i.point chart;.

Limit the use of .i.strip chart; and .i.trend graph;s due to their CPU and graphical

requirements. Usually you can substitute a line graph for one of these. Refer to Performance Issues in this section for more information concerning the performance of these data formatters.

.c.Color Dynamics;

.i.Color dynamics; allow you to change the color of a graphical object based on the value of a data source variable. This is convenient to show the value of a variable when low precision is desirable such as on/off states or where there is only a few important states for a variable.

Color dynamics cannot be imported into a drawing, since all of the data source variables are stripped from the included file.

If you require color changes on an imported display, you must construct multiple subdrawings with each .i.subdrawing; having a single color. Then as the dynamic drawing changes the subdrawing, the color will appear to change. Refer to Subdrawing Dynamics.

.c.Subdrawing Dynamics;

Drawing dynamics are used to .i.animate displays; or to make color changes that cannot be done using the normal color dynamics. Drawing dynamics are created by assigning multiple subdrawings to be displayed at a single location on a display based on the value of a data source variable. When the value of the data source variable changes, CRISPdraw changes the subdrawing. By cycling through a number of values, the display can appear to be animated in a more complex manner than can be done using only color table animation.

Subdrawing dynamics can also be used to create complex color changes that are dependent upon only a single data source variable.

.c.Set Guidelines;

.i.Guidelines; help you make your display system more consistent, thus making it easier for your user to understand and use. This is especially true when a user is venturing into a new area of your display system. If the new area is consistent with the old familiar area, your user will feel at home. If the areas are inconsistent, the user must learn and remember two ways of working with your displays. They must also remember when to switch ways of using your displays. This needlessly burdens your users.

Document your .i.conventions; so persons making changes to your displays will not have to discover your conventions or create their own; they can make changes consistent with your original intent. Writing your conventions will force you to express your ideas concisely and will guide you while creating displays.

.c.Coding Guidelines;

Coding enhances the recognition and interpretation of data elements on a display by directing the user's attention to the most noticeable elements.

(Continued on next page.)

Coding Guidelines (cont)

.i.Display data levels; Display data can be categorized into 5 levels. Level 1 data should be the most noticeable elements in a display. This level includes .i.alarms;, .i.mode indicators;, and .i.data quality indicators;. Level 2 includes ordinary .i.process control data; and should be slightly less noticeable. This level may be subdivided into primary and supporting data items. Level 3 elements convey referential information, such as bounds. Level 4 elements include markers such as scales, units and labels. Level 5 elements are connectors, group markers and background images.

Adjusting an .i.object;'s dynamics, color, size and intensity changes its .i.object visibility;visibility relative to other objects on the display. A user sees blinking or moving objects before any others; a yellow, orange or yellow-green object before a brown or blue object; a large object before a small object; or a bright object before a dark object.

Applying these principles to designing displays means that .i.alarms:active;active alarms should be big, bright and blinking while .i.alarms:inactive;inactive alarms should be almost invisible, that is, dark and static. .i.process control data:primary;Primary process control data should be big and bright. .i.process control data:supporting;Supporting process control data should be smaller and not so bright. Scales, labels and units should be darker. Background images, grouping markers and connecting lines should be quite dark, since the images and grouping markers (such as rectangles) tend to be large.

.c.Usage Standards;

Usage .i.Standards:usage;standards improve the consistency of your display system and make it easier to use. The following are some ideas on developing usage standards for operator displays.

.c.Information A User Must Enter;When designing your display system, if possible, arrange the text that users must type in so that it only changes at the end of the string. This reduces the amount of deleting and retyping that will be required of users and minimizes operator effort.

.c.Information/Symbols To Appear On Each Display

Most display systems have some information that will appear on each display in the system. These items usually include navigational aids, labels, and icons. You can make these into .i.subdrawing;s and merge them into your display under development. Refer to Subdrawings in Planning a Display System section of this manual and the Drawing within the Complex Editing subsection of the General Editing section in the *Window Workstation Reference Manual* for further information about subdrawings. Refer to Merging View within the Using the Commands Menu subsection within the General Editing section in the *Window Workstation Reference Manual* for further information about merging views.

.c.Previous/Next Buttons; Any button used on multiple displays should be placed at the same location, especially

when it is used as a `.i.button:previous;previous` or `.i.button:next;next` button. This minimizes a user's effort when paging through the display system.

- .c. **Go To Buttons**; You may want to give **Go To** buttons a unique color to indicate they perform an action when the user selects them.

When you create a **Go To** button, give the same run-mode action to the text string contained in the button as you do to the rectangle that comprises the button. This eliminates problems caused by users selecting the button, but the button does not do anything.

.c.Function Key Usage; .i.Function keys:advantages;Function keys free up valuable display space and they are useful in environments that discourage the use of pointing devices such as a mouse. They have the .i.Function keys:disadvantages;disadvantage of requiring the user to remember which function key performs a given function, although you can provide the user with a map to the function keys as a part of the display.

The CRISPdraw function key design allows both .i.Function keys:global;global and local key definitions. Prudent design prescribes designing your displays so local keys have similar functions on each display. For example, if the **F6** key switches displays to the next key in a series, the **F6** key should always be used as the next key. You can also create a global function key binding for the **F6** key that will switch displays to a known display. The point is to always have the **F6** key always switch displays, not switch displays for some displays and set CRISP variables on other displays. Refer to Function Keys in the *Window Workstation Reference Manual*.

The file .i.CRISPDRAW_GBL_FKEYS.DAT;CRISPDRAW_GBL_FKEYS.DAT contains the global function key definitions.

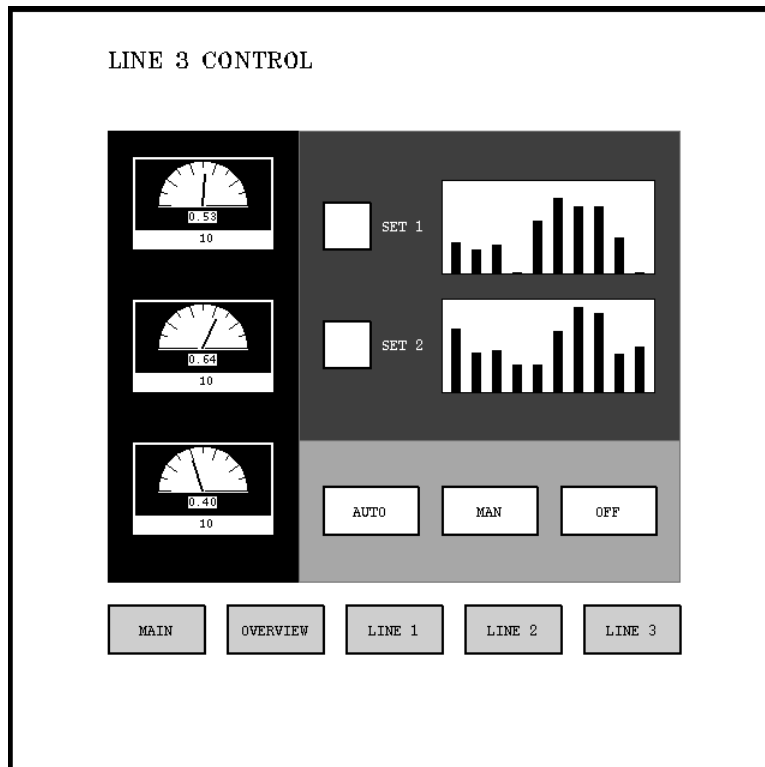
.c.Display Structure; The major purpose of structuring your displays is to assist your users in locating important information on a given display and discriminating between important and supporting information.

.c.Grouping Graphical Elements; Group Graphical elements together to emphasize their relationships. This can be done to indicate the following: most important elements in a display, the elements for each step of a procedure if more than one step is represented on a display, or the variables related to a portion of the process being controlled.

.c.Grouping Methods; There are several means of grouping elements together. Use background color, lines, size, or shape to group particular elements of a display. Each of these are described in more detail in the following.

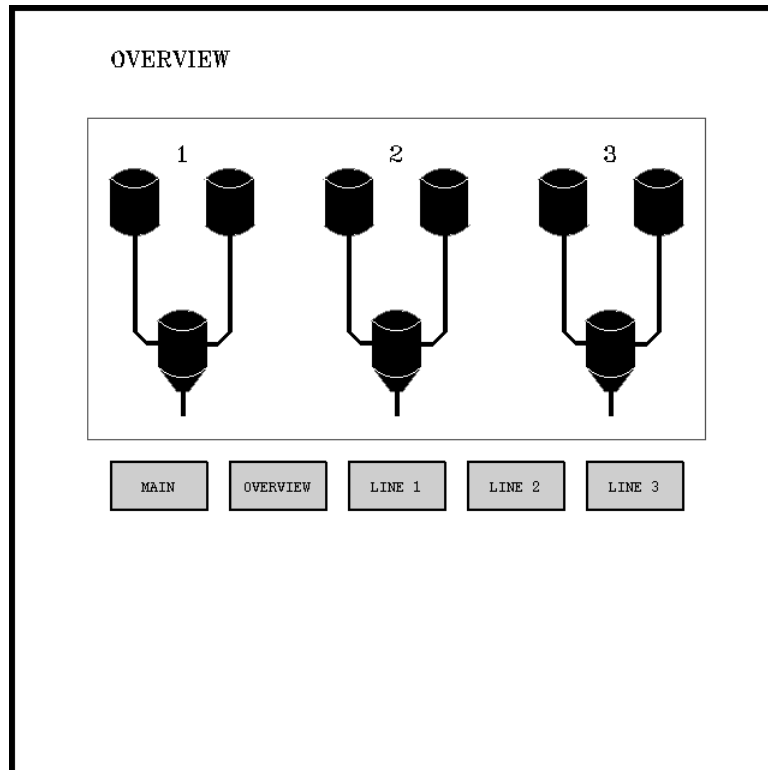
- c. **Background Color;** Background areas of a display can be used to show groupings. This is done by creating a filled rectangle or polygon encompassing the elements to be grouped. The color of the rectangle or polygon should be selected to blend into the background of the display. In general, it is better to turn the edge attribute off, since defining the edge of the grouping creates a visual barrier.

You can also use this to create several groups within a larger group. Colors for these larger groups should be intermediate between the color of the individual group and the background color. Figure 6 illustrates this concept.



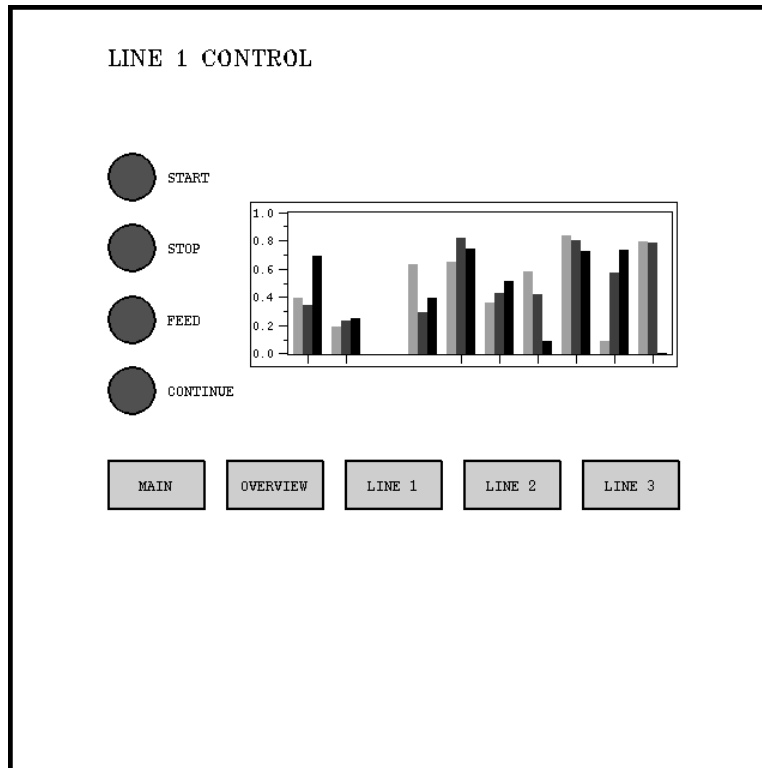
c. Figure 6. Background Colors

- .c.**Lines;** Use lines to connect related elements or to show the flow of material from one processing step to another. Lines give the user a visual trail to follow from one graphical element to another, although they can be difficult to follow when a large number of them converge in an area of the display. Figure 7 illustrates this concept.



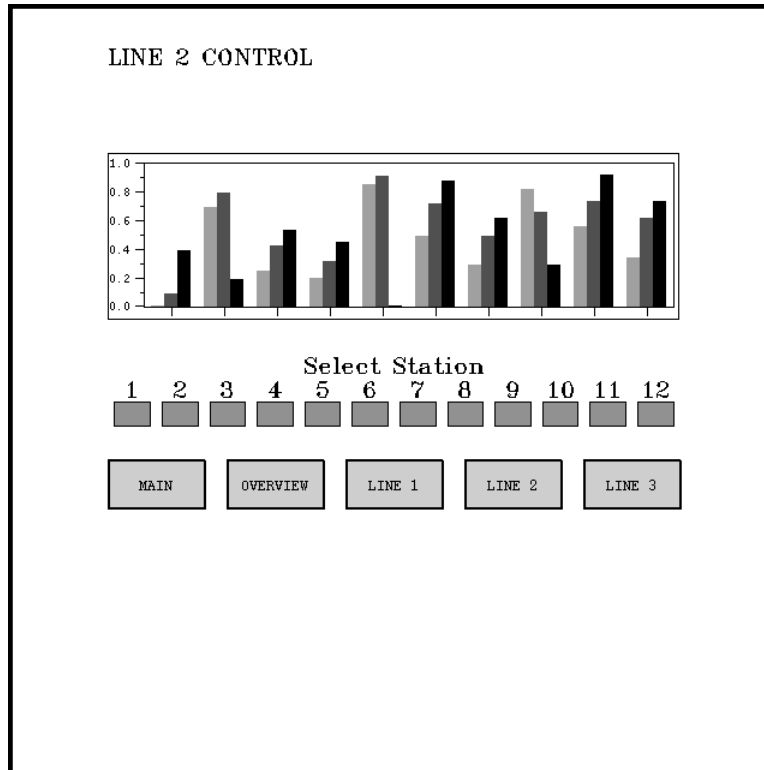
.c.**Figure 7. Line Grouping of Graphical Elements**

- .c.**Shape**; Shape can be used to group elements by giving related elements similar shapes. This is effective when other constraints require the elements to be located apart from each other. This can be applied to both foreground elements as well as background elements. Figure 8 illustrates this concept.



.c.**Figure 8. Shape Grouping of Graphical Elements**

.c.**Size**; Use size to group elements together, especially when other constraints require the elements to be located apart from each other. Figure 9 illustrates this concept.

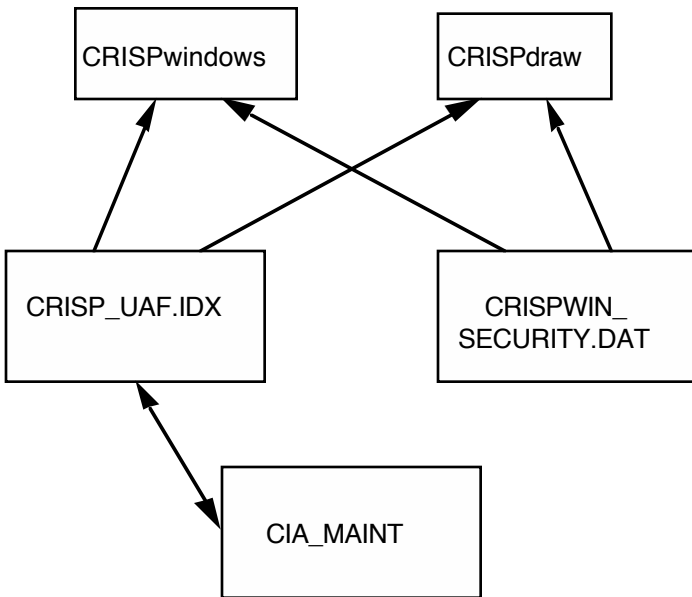


.c.**Figure 9. Size Grouping of Graphical Elements**

.c.**Security**;

The Window Workstation security allows a system administrator, who may or may not be the same as the OpenVMS system administrator, to restrict access to objects in a display system. The system administrator can restrict access to individual CRISP symbols, display files and Window Workstation functions.

.c.Security System Design; There are three major components to the Window Workstation security system: the user database, the symbol database and the executable programs. See Figure 10.



.c.Figure 10. Security System Files

User Database

The first component is the `.i.security:user database;user database`, `CRISP$CFG:CRISP_UAF`. For each user, it contains a `.i.Username;username`, an encoded `.i.Password;password`, the `.i.Security privileges;security privileges`, the `.i.System privileges;system privileges`, and the `.i.User-defined privileges;user-defined privileges`. The username and password correspond to the usual notions of what these items are. The `.i.security; privileges` permit a user to modify the user database. The system privileges permit a user to edit a Window Workstation display, print a hardcopy, exit Window Workstation, or start a `CRISPdraw` process from within `CRISPwindows`.

The system administrator can set Window Workstation privileges for each user. There are 32 of these privileges. The names of the privileges are kept in file `CRISP$CFG: CRISPWIN_PRIV.NAMES`.

Central Administration

If the CRISP Central Administration product is installed on a central machine, all the Window Workstations will be in a single security domain. The `CRISP_UAF` file can then be managed on any machine in the domain.

The master `CRISP_UAF` file is accessed via DECnet. If the network is not available, a local copy is used, but is locked against any changes. For each process that uses the `CRISP_UAF` file (`CRISPwindows`, `CRISPdraw`, `CIA_MIAANT`, etc.), a DECnet task-to-task process called `CENTRAL_OKAY` is started. `CENTRAL_OKAY` communicates with the central machine, and maintains system logical name `CRISP$CENTRAL_OKAY`.

Symbol Database

The second component of the security system is the `.i.security;symbol` database;symbol database. This file contains a default mask and a mask for each symbol whose security requirements differ from the default. When the user tries to access a symbol, Window Workstation retrieves its mask and combines it with the user-defined mask of the current user to determine if the user is allowed access to the symbol. If the user is permitted access, execution continues but, if the user is not permitted access, an error message is printed. The symbol for a file is the file name as specified in the **GOTO** command. The symbol for a CRISP variable is the `NODE::DATABASE:VARIABLE` name given in the **MODIFY** or **TOGGLE** command.

Database Administration

The third major component is the executable programs that maintain and use these databases. The `CIA_MAINT_WINDOW` program is used to maintain the user database. You must use a workstation with the X Window system or an X terminal to run this program, since it uses a graphical user interface. A companion program, `CIA_MAINT`, is used when the system administrator does not have access to a windowing terminal or a workstation.

If the Enhanced Security product license (CRISP-WWSPLUS) is installed, the `CIA_MAINT` programs and the Window Workstation enforce extra security requirements.

Caution: If the WWS privilege names are modified on any node other than the Central Administration node (if any), file `CRISP$CFG:CRISPWIN_PRIV.NAMES` must be copied manually from that node to the central node. Otherwise, other systems will not see the new names, although privileges will work properly.

Privilege Masks

Two algorithms determine whether a user can access a given symbol. Algorithm 1 simply ANDs the user mask with the symbol's mask. If the result is non-zero, then the user can access the symbol. Algorithm 2 is more complicated since it ANDs the user mask with the symbol mask and then it XORs the result with the symbol mask. The result must be zero for the user to be granted access to the symbol. Refer to Workstation Security in the *Window Workstation Reference Manual*.

If you want to run Window Workstation without `.i.security;` enabled, which is not recommended, you can do this easily by using one of the following values as the `.i.default` value; in the `CRISPWIN_SECURITY` file, depending on the algorithm used. If you use Algorithm 1, use `FFFFFFFF` as the default value and set at least one of the bits in the user privilege mask. If you use Algorithm 2, use `00000000` as the default value and clear all of the bits in the user mask.

These two algorithms allow you to tailor the meanings of the individual bits in the masks. With Algorithm 1, if the user and symbol masks have ANY matching bits, permission is granted. With Algorithm 2, if the user mask has all of the bits in the symbols mask turned on, permission is granted. The difference between the two algorithms is illustrated by the following examples. These example use only four bits instead of the 32 bits that are available. Rather than present the numbers as hexadecimal values, they are presented using a binary representation.

Algorithm 1

This first example illustrates how to define user and symbol masks using Algorithm 1. The following scenario assumes there are two identical production lines with two sets of operators. Operators should set only those variables that control the line they are assigned to. Supervisors may set variables on either line plus some other variables, such as recipe management. Supervisors may not set any line maintenance variables. Engineers may set variables on either line plus line maintenance variables. Engineers should not be able to set the recipe management variables.

The following privileges are to be protected.

Maintenance	1000
Recipe Download	0100
Line 1 operation	0010
Line 2 operation	0001

Assume we have the following users and their masks.

Engineer	1011
Supervisor	0111
Operator1	0010
Operator2	0001

The following symbols will then have the following masks.

l1dsp:maint.v	1000
l2dsp:maint.v	1000
line1::db:load_recipe	0100
line1::db:start	0010
line1::db:stop	0010
line2::db:load_recipe	0100
line2::db:start	0001
line2::db:stop	0001

The engineer can access either of the maint.v displays and the start and stop variables for either line. The engineer cannot access the load_recipe variable. The supervisor can access the load_recipe, start, and stop variables, but not the maint.v display. The operators can access only the start and stop variables. Sample calculations are listed in the following table.

Symbol	User	Sym Mask	U Mask	Result Access
l1dsp:maint.v	Engineer	1000	1011	1000 y
l1dsp:maint.v	Supervisor	1000	0111	0000 n
l1dsp:maint.v	Operator1	1000	0010	0000 n
l1dsp:maint.v	Operator2	1000	0001	0000 n
line1::db:start	Engineer	0010	1011	0010 y
line1::db:start	Supervisor	0010	0111	0010 y
line1::db:start	Operator1	0010	0010	0010 y
line1::db:start	Operator2	0010	0001	0000 n

line1::db:load_recipe	Engineer	0100	1011	0000	n
line1::db:load_recipe	Supervisor	0100	0111	0100	y
line1::db:load_recipe	Operator1	0100	0010	0000	n
line1::db:load_recipe	Operator2	0100	0001	0000	n

Algorithm 2

The second example illustrates the use of Algorithm 2 and assumes there are two supervisors, one for each line. We then have the following users and their masks.

Engineer	1011
Supervisor1	0110
Supervisor2	0101
Operator1	0010
Operator2	0001

The following symbols will then have the following masks.

l1dsp:maint.v	1000
l2dsp:maint.v	1000
line1::db:load_recipe	0110
line1::db:start	0010
line1::db:stop	0010
line2::db:load_recipe	0101
line2::db:start	0001
line2::db:stop	0001

Supervisor1 can not modify the load_recipe variable on line 2 and Supervisor2 can not modify the load_recipe variable on line 1. Sample calculations are as follows.

Symbol	User	Sym Mask	U Mask	Result Access
l1dsp:maint.v	Engineer	1000	1011	0000 y
l1dsp:maint.v	Supervisor1	1000	0110	1000 n
l1dsp:maint.v	Supervisor2	1000	0101	1000 n
l1dsp:maint.v	Operator1	1000	0010	1000 n
l1dsp:maint.v	Operator2	1000	0001	1000 n
line1::db:start	Engineer	0010	1011	0000 y
line1::db:start	Supervisor1	0010	0110	0000 y
line1::db:start	Supervisor2	0010	0101	0010 n
line1::db:start	Operator1	0010	0010	0000 y
line1::db:start	Operator2	0010	0001	0010 n
line1::db:load_recipe	Engineer	0110	1011	0100 n
line1::db:load_recipe	Supervisor1	0110	0110	0000 y
line1::db:load_recipe	Supervisor2	0110	0101	0010 n
line1::db:load_recipe	Operator1	0110	0010	0100 n
line1::db:load_recipe	Operator2	0110	0001	0111 n

Bias Bit Behavior

The privilege mask behavior described above is modified if logical name

CRISP\$WWS_BIAS_BIT is defined, and is a number greater than zero. This is known as “bias bit”.

When the bias bit is set, the user’s privilege mask is right-shifted by that number of bits before any operations are performed. For example, a user’s mask of 1001101 becomes 0010011 if the bias bit value is 2.

The bias bit feature allows the WWS privilege mask to be sub-divided into several sub-domains. For example, if users are to have two different set of privileges on two different groups of systems, the bias bit value on one set of machines can be zero, and 16 on the other set. If the symbol masks have the high 16 bits set or clear (depending on the algorithm in use), the privilege masks have been divided into two sub-domains.

In actual practice, it is recommended that some bits be reserved for future expansion. In the case of just 2 sets of systems, bias bit values of 0 and 4 might be more appropriate.

.c.Designing User Security; The CRISP security system allows you to assign up to 32 different .i.security:privileges; to each user.

.c.Assigning User Privileges; Each different capability that you wish to protect will require a single bit in the user and symbol masks. You can protect up to 32 different capabilities, subject to the bias bit feature described above.

.c.Securing CRISP Variables; Each secured CRISP variable must have an entry in the .i.CRISPWIN_SECURITY file;CRISPWIN_SECURITY. This entry consists of the symbol mask and the name of the CRISP variable being secured. The name must exactly match, except for the case of the characters, the name used in the **SET** or **TOGGLE** run-mode action.

.c.Securing Displays; Each secured CRISPdraw display must have an entry in the .i.CRISPWIN_SECURITY file;CRISPWIN_SECURITY.DAT file. This entry consists of the symbol mask and the name of the display file being secured. The name must exactly match, except for the case of the characters, the name used in the **GOTO** run-mode action.

.c.Adding Users To The Security Database

Users are added to the .i.CRISP_UAF database;CRISP_UAF database using the CIA_MAINT_WINDOW program. This program will lead you through the task of adding a user or modifying existing user information.

.c. Performance Issues; Several issues should be considered when fast display update or display switching is required for an application.

.c. Display Switching Times; Large display caches and simple displays yield the best .i. display switching times;. Display caching speeds up the switching times by eliminating the need to read a display file from disk, but it increases virtual memory requirements of CRISPdraw.

The number of graphical elements and graphs largely determine the display switching times. Although a display consisting mostly of rows and columns of digit graphs may look rather plain and simple, it is a quite complicated assembly of graphs that can take a considerable amount of time to switch.

If the CRISP Central Administration product is in use, display switching times can be negatively affected on a very slow network, or if the central system is overloaded.

.c. System Resource Usage From Data Graphics

Some display formatters consume larger amounts of CPU than others. .i. Strip chart;s and .i. trend graph;s have especially high CPU requirements.

You can reduce the amount of CPU required for updating .i. line graphs; and related graphs when displaying arrays by setting the graph increment to the same number as the number of elements in the array. Set Samples, Scroll By, and Increment in the Edit Graph menu and set the Vector length in the Edit Variable menu. Refer to Samples, Scroll By, and Increment in the Edit Graph menu and set the Vector length in the Edit Variable menu in the *Window Workstation Reference Manual*.

.c. Printing Issues;

The intended use of hardcopies will influence the requirements placed on displays. If accurate color rendition is important, test your displays and adjust the screen colors for best results.

The hardcopy process has several steps. The first step is to get an image of the screen from the X server. This image is then written to a file and a subprocess is spawned to process the file. The subprocess reads the file, translates the X image into a printer specific format, writes the translation to a file, and then copies this file to the desired printer.

The printer currently supported is the Digital Equipment Corporation LJ250 printer. It uses an ink jet having a resolution of 180 dots per inch in each of cyan, magenta, yellow and black colors. Only the cyan, magenta, and yellow colors are specified when a color copy is created. The black ink usage is generated internally by the printer.

Refer to the SCRPRINT.COM section in the *Window Workstation Reference Manual* for detailed information about using this program.

.c. **Optimum Screen Size;** The .i. optimum screen size; for the LJ250 is 900 pixels wide by 720 pixels high.
This creates a print that is 10 inches wide by 8 inches high.
(Continued on next page.)

Optimum Screen Size (cont) Each pixel on the screen will be represented by four dots on the paper giving an effective resolution of 90 dots per inch with five levels per color. A dithering algorithm is used to eliminate aliasing artifacts from the printed copy and to give a more pleasing appearance to the image.

If the display size is larger than the optimum or less than 75% of the optimum size, a scaling algorithm is used to expand or shrink the image as necessary. This scaling algorithm will introduce color gradations in areas near color changes. These color changes adversely affect the clarity of text or of regions where the size of the colored area is small.

.c.Colors Displayed On The Screen vs. Colors On Paper

The colors displayed on the paper can be significantly different from the colors displayed on the workstation screen. There are a number of reasons for this. Most workstation graphics systems are capable of displaying 256 levels each of red, green, and blue whereas the printer is capable of reproducing only 2 levels of cyan, magenta, and yellow. The human eye responds differently to the luminous colors of the workstation than it does to the reflective colors on paper. Finally, the mapping of the red-green-blue color system of the workstation to the cyan-magenta-yellow color system of the printer is not exact, especially when ink and jet variations are considered.

If accurate .i.color rendition; is important, test your displays and adjust the screen colors for best results. The primary (cyan, magenta, yellow, black and white) colors will reproduce most accurately, especially if they are fully saturated.

.c.Time To Print;

The time to print a display on the LJ250 varies from 5 to 15 minutes or more. The largest portion of this time is usually used to communicate the image from the computer to the printer using a serial line running at 9600 baud. At this speed, each block in the file takes about one-half second to send, so a 1000 block file requires about 500 seconds or just over 8 minutes.

Translating the image into the printer format can also require significant processing, especially if the image must be scaled and the colors dithered. The use of non-primary colors in large areas will also increase the processing required since each pixel must be dithered. This will also increase the size of the file significantly since the printer file uses a run-length-encoded format. The dithering process randomizes the output and reduces the probability that a long run of the same color will occur.

If the processor of your machine is fully loaded, print processing will be slowed since this is done as a batch job at a lower priority than interactive processes.

Generating a black and white image speeds up the printing process since only one color channel is generated and the file will be about one-third the size of the color image.

.c.Disk Usage;

Each display image file can contain approximately 2600 blocks of data. The

printer format file can be up to approximately 2500 blocks in size. Creating black and white images reduces the printer format file size by about two thirds.

.c. **Detail Design Checklist;** Checklist to aid the user in identifying important design issues.

- Historian Access CRISP logic
- Guidelines documented
 - Display data coding defined
 - Usage standards in place
 - Function key usage standardized
- Display structure defined
- Security system defined
 - Defined CRISP variables to be secured
 - Defined displays to be secured
 - Privilege bits defined
- Performance issues addressed
- Printing issues addressed
 - Screen size set for printing
 - Colors tested for optimum rendition

.c.General;

By now, you should have a concrete idea of the appearance and content of each display and how it relates to other displays. During this stage you will create the actual displays using CRISPdraw. If you have most of the earlier issues under control, this stage will go smoothly. If there are many unresolved issues, you may find yourself continually making major changes to the displays and falling farther and farther behind your schedule. View this as a sign that you need to revisit your prototypes or detailed design and resolve the remaining issues. This section focuses on tips that will simplify creating displays.

.c.Symbol Libraries;

Symbol libraries can save you much work when properly used, but you should be aware of their limitations before you start. .i.Symbol libraries; are typically defined as a series of subdrawings and will have all of the limitations of .i.subdrawing;s. This means the subdrawing may not contain any dynamics. Refer to Subdrawings in Planning a Display System section of this manual and the Drawing in the Complex Editing subsection with the General Editing section in the *Window Workstation Reference Manual* for further information about subdrawings. Refer to Merging View within the Using the Commands Menu subsection in the General Editing section in the *Window Workstation Reference Manual* for further information about merging views.

.c.Drawing Usage;

Use .i.logical names; in display names in **GOTO** run-mode actions. This make it easy to move the displays to a new directory or to a new node if necessary. For instance, rather than have the run-mode action "GOTO CRISP\$DEVICE:[CRISP.WWS.DSP]LINE1.V" use "GOTO LINE1_DSP:LINE1.V". If you must move the line 1 displays in the future, all you will have to do is redefine the logical name for the directory.

.c.Displaying Data;

.c.CRISP Data Sources Tips;

.c.WORF Alias File; The .i.WORF alias file;, CRISPPWIN_WORF_ALIAS.DAT, lets you define symbolic names for nodes when creating data sources. This is helpful when you are creating the displays on one system and deploying the displays on another system, when you do not know the eventual location of the database, or when you are creating displays to be deployed on many machines. It also gives you the flexibility to easily move databases to other machines.

.c.Create Template Displays; You can reduce the effort required to create displays by creating and using .i.display template;s. These are skeletal displays that you complete by adding elements. Consider adding a rectangle to the template display having the name ".i.draw.area;" (refer to Defining the Display Area subsection in the *Window Workstation Reference Manual*). The "draw.area" rectangle creates an area that is guaranteed to be visible on the display screen regardless of the aspect ratio of the CRISPdraw window.

You should also put all of your commonly used objects in the display template. It is easier to delete objects from the display than it is to create additional objects.

.c.Create Blank Display With DSLs; If most of the displays in a system use a similar .i.data source list;, you can reduce your effort by creating a blank display containing the data source list with all of its variables. You can then merge this view into individual displays and delete the variables that are not used.

If you have multiple data source lists, you can create a display for each data source list and merge the ones you need for a particular display.

.c.Create Templates Of Common Elements

You can save much time by creating a .i.template variable; in a .i.data source list; and copying it when you need a new variable. You do not need to modify the variable type since the type will be determined when the data source list is resolved.

You can also use this method for more general graphical objects that appear on numerous displays. You can create a subdrawing containing the graphical objects, merge the display into your new display, and delete the objects that are not used on your new display.

.c.HISTORIAN Access; Window Workstation cannot directly access data stored in .i.CRISP HISTORIAN; files. However, you can access .i.HISTORIAN data; HISTORIAN data by using a .i.CRISP logic; to get the data, normalize it, and deposit it in an array in a CRISP database. Once the data is in the database, Window Workstation can access the data as a .i.CRISP real-time array; variable. The data can then be displayed in any of the CRISPdraw graph types suitable for array data.

Historian data can also be displayed with the Historian Viewer layered product.

A CRISP function call, .i.Function Call:SCAN_HIST;SCAN_HIST, provides a flexible means of accessing .i.HISTORIAN data; HISTORIAN data and putting it in a CRISP database. You can specify the interval over which to sample the recorded data. You can specify a value to search for to start the interval. You can specify the frequency at which the data should be sampled. You trigger the SCAN_HIST call by causing a trigger variable to change state. The SCAN_HIST call returns an array of sampled data and an array of time stamp strings showing the time of each data sample. A set of parameters can be passed to the SCAN_HIST call to provide the sampling function you need for your application. Refer to the CRISP/32 Function Calls Reference Manual for a complete description of the SCAN_HIST call.

Once SCAN_HIST is called, the logic that made the call is blocked while SCAN_HIST runs to completion. Put your SCAN_HIST calls in a separate logic that does not do any control functions to avoid having time delays in your control system.

The data you retrieve using SCAN_HIST will have the same range as the real-time variable that the data was originally captured from. If you need to show different variables with different ranges in a single graph, normalize the data in your CRISP logic using SCALE, a CRISP math function call. You may choose to normalize the data to a 0 to 1 range, or to a 0% to 100% range, or to some other range suited to your application. The .i.Function Call:SCALE;SCALE function should take the array of data retrieved by SCAN_HIST and output a normalized array.

.c.Accessing And Displaying HISTORIAN Data

Several of the parameters used by the `.i` function call:SCAN_HIST; call should be set up interactively by the operator. These parameters can be put into CRISP real-time variables for use by the SCAN_HIST call. Create a setup screen on which the operator enters values for the required parameters. There could also be a pushbutton on this setup screen the operator would use to trigger the SCAN_HIST after all the parameters have been entered.

You also need to create a display for viewing the `.i`.HISTORIAN data;HISTORIAN data. This may be a view file that displays the array of normalized data in a `.i`.line graph;, `.i`.bar chart;, or other suitable graph type. Refer to the *Window Workstation Reference Manual* for details on displaying array data in graphs.

Since the data you display may be a very large array, it will probably be difficult for an operator to read exact values of data from a graph. Your display should provide some means of viewing individual data points in the array. One way to do this is to provide a means of indexing into the array to obtain single data points and provide a marker in the graph to show which individual point you are indexing. The following example explains how to do this.

Step 1. In your `.i`.CRISP logic;, declare an array for the raw `.i`.HISTORIAN data;HISTORIAN data, an array for the HISTORIAN time stamp text strings (we will call this `time_stamp_array`), an array for the normalized HISTORIAN data (we will call this `norm_hist_array`), and an array of floats called `marker`. All these arrays should be the same size (we will call this array size `N`). Also declare a Numeric variable, called `index`, and a logical variable called `trigger`.

Initialize the value of `index` to `N-1`, so that, initially, this variable will index the last member of the arrays. Initialize all members of the `marker` array to zero. Initialize `trigger` to 0.

Step 2. Create logic statements that perform the following.

- a. Reads the current value of `index`. Clears `marker(i)`, where `i` is all numbers 0 through `N-1`, except `i = index`.
- b. Sets the value of `marker(index)` to the top of the normalized range of your data (for instance, if you've normalized your data to a range of 0 to 100, set `marker(index)` to 100.0).
- c. Reads and records the value of `trigger`. If `trigger` changes from 0 to 1, clear `trigger` to 0 and trigger the `.i`.Function Call:SCAN_HIST;SCAN_HIST call. When SCAN_HIST returns, normalize the raw HISTORIAN data and put the normalized data into `norm_hist_array`.

Step 3. Create a view file with a Bar-Line graph. The first variable in a Bar-Line graph is plotted as a bar chart. All other variables are

plotted as lines. Select the `marker` array as the first variable in the graph. Link the array of normalized HISTORIAN data to the graph. Thus, the `marker` array will look like one vertical line on the graph against the line graph that displays your normalized HISTORIAN data.

Step 4. Create a Digits graph linked to the CRISP variable `index` with a run mode action that allows the operator to modify the `index` variable.

Step 5. Create a Text graph linked to the variable `time_stamp_array(index)` and a Digits graph linked to the variable `norm_hist_array(index)`.

Step 6. Create a pushbutton with a run mode action that allows the operator to toggle the `trigger` variable.

When the logic is running and you bring up this view, selecting the pushbutton created in Step 6 will cause the `.i.HISTORIAN` data; HISTORIAN data to be obtained, normalized, and displayed on the `.i.line` graph;. Selecting the `.i.Digits` graph; created in Step 4 allows setting the value of the `index` variable. When `index` is set to a new value, the marker bar in the graph created in Step 3 will be positioned to indicate an element in `norm_hist_array` and the Text and Digits graphs created in Step 5 will display the time stamp and value of the element indicated by the marker bar.

The marker bar functions as a pseudo scrollbar. It can be made a little fancier by creating additional push buttons that cause the logic to increment and decrement the value of `index`, so that the scrollbar can be adjusted back and forth on the graph.

.c.CRISP Data Source Variables; When you create a `.i.CRISP` data source variable;, it is not necessary to define its type since this will be determined when the variable is resolved.

When you add a CRISP data source variable, it uses the last variable in the data source list as a template for the new variable. You can use this to create a `.i.template` variable; that is used to create a number of variables. This is especially useful when several variables are created that differ only in the endings of their names.

