

```

% -----
% Copyright (c) 1990-1992, Eurotherm Limited
% File: cworks.tex
% Author: Jenny Oliver <jm00> and Mike Dillamore <md00>
% Date: Wed Jun 3 11:00:23 BST 1992
% Version: 1.0
% Content: ControlWorks Functional Specification
%
% $_Revision
%
% THIS FILE IS PRESENTLY OUT FOR EDITING.
% :--1.1 1.2 jo00 92/06/24 14:48:29
%
% WARNING: Any editing between above marker line and end of
% rev. history marker line will be lost at next change.
%
% 1.1 92/06/24 14:48:21 jo00
% date and time created 92/06/24 14:48:21 by jo00
%
% $_End
% -----

\documentstyle[eispec,epsf]{article}

\newcommand{\epsffig}[3]{
  \begin{figure}[tbp]
    \centerline{\epsffile{#1}}
    \caption{#2}
    \label{#3}
  \end{figure}
}

\def\startchange{\begin{center}$\Downarrow\hspace{1in}\Downarrow\hspace{1in}\Dow
\def\endchange{\begin{center}$\Uparrow\hspace{1in}\Uparrow\hspace{1in}\Uparrow$}

\begin{document}

\project{Group Configuration Tool}
\docname{Product Specification}
\author{GCT Team}

\docno{HP024674}
\revision{2}

\doctype{Product Specification}
\controllingdept{ENGINEERING}
\originatingdept{ENGINEERING}

\signatoryone{Project Manager}
\signatorytwo{Technical Director}
\signatorythree{Marketing Director}

  ectronicpath{wren:/wren2/gct/doc/specs/gct/_spec/gct/_spec.tex}
\stampbox

\begin{distribution}
\for {Project File}
%\for {Ian Hughes}

```

```
%\for {Keith Jones}
%\for {John Juer}
%\for {Bob Lewis}
    .or {Brian Smith}
\end{distribution}
```

```
\controlsheets
```

```
\begin{history}
\rev{A} {June 24, 1992} {Pre-release copy}
\rev{1} {Sept, 1992} {Version 1}
\rev{2} {March, 1993} {Added detail to sections on views and
editors ({\S}\ref{viewsgct}{ff}); Updated to reflect changes since
project conception}
%Remember to update \revision{n} when new revision added
\end{history}
```

```
%[ Editorial Note 1:] Areas containing changes since the previous
%version are delineated by lines containing \startchange and \endchange.
```

```
[ Editorial Note:] Issues that require resolution are denoted [{\em thus}].
\tableofcontents
\listoffigures
\pagebreak
```

```
\part{Project Overview}
\section{Scope}
```

This document is the product specification of the Group Configuration Tool (GCT), a software tool providing a highly interactive, multi-windowing, graphical interface for the creation and management of control applications. This specification gives an introduction to the control model supported by GCT, outlines some architectural considerations and provides an overview of how a user will interact with the system.

```
\subsection*{Glossary Of Terms}
```

```
\begin{itemize}
```

```
\item {\bf IEC 1131} - The draft international standard
''Programmable Controllers''. The term IEC 1131 is usually taken to
refer to part 3 of the standard : ''Programming Languages''.
```

```
\item {\bf CDL} - Configuration Definition Language used to specify all aspects
of a distributed control strategy. CDL is an subset~/ superset of the IEC
1131 ST language.
```

```
\item {\bf FBD} - Function Block Diagram graphical representation of
continuous control strategy.
```

```
\item {\bf SFC} - Sequential Function Chart graphical representation of
sequential control strategy.
```

```
\item {\bf ST} - Structured Text textual language to specify all
aspects of control strategy.
```

```
\item {\bf Application Entities} - the CDL components that make up an
application.
```

\item {\bf Block} - a Block is the unit of definition. That is to say, it is an Application Entity that can be saved in a library, moved between libraries, deleted from a library, etc.. A block may be a Configuration, Resource, Program, Function or Function Block.

\item {\bf Sub-Block} - a part of a Block that can be edited or viewed using its own editing paradigm. For example, the body Sub-Block of a Function Block may be edited using an SFC whereas its declaration Sub-Block may be edited by the template editor. A Sub-Block may be a Transition, Step, Action, cold start or Wiring expression, variable declaration, block body, etc..

\item {\bf POU} - A Program Organisation Unit is the collective IEC 1131 term for a Program, Function or Function Block. That is to say, a POU is a block which is a sub-unit of a control strategy. A POU has a type definition that can be instantiated as many times as required within a control strategy.

\end{itemize}

\begin{relateddocs}{1cm}

|                    |   |
|--------------------|---|
| \bibitem{IECSpec}  | IEC DIS 1131-3 (Programming Languages for Programmable Control) |
| \bibitem{CDLSpec}  | CDL Specification (HP024674C307), J.W. Juer                     |
| \bibitem{ResSpec}  | HP024105: Resource Manager Release 1 Specification, J.W. Juer   |
| \bibitem{IECJuer}  | IEC65A Control Languages - A Practical View, J. Juer and I.P.   |
| \bibitem{GAPESpec} | GAPE Preliminary Specification, M.S. Dillamore and J.J. Olive   |
| \bibitem{ResOver}  | An Overview Of The Resource, I.P. Hughes                        |

\end{relateddocs}

\section{Aims Of The Project}

\subsection{Development model}

The architecture of GCT is designed to be open and flexible so that it is easy to add new editors, and it is easy to have teams of engineers working independently and concurrently doing so.

The development plan will take advantage of this by following an incremental evolutionary development model, in order to get immediate user feedback and hopefully payback for the development. The aim will be to produce releases of the product every 3 to 6 months. In order to achieve this each release will obviously not contain all the features desired by marketing; discussion will have to occur to decide which new features to add on the basis of the cost of adding them against the payback. In addition any feature added must be reviewed against its contribution to and compatibility with the overall goals of the project. Aiming for such a development cycle will also ensure flexibility in planning releases --- if a new feature is needed gently it can be added in more easily to a non-monolithic plan that caters for frequent releases.

\subsection{Release Schedule}

A provisional release schedule, based on the current project team, is as follows ---

```
gin{minipage}{7in}
\begin{tabular}{l|p{0.9in}|p{2.9in}|p{1.2in}}
```

Approx Date & Version & Features added & Targets added \\ \hline

{November '92} & {Demonstrator} \footnote{This version was demonstrated at the Core Technical Conference, Leonardslee, 14th December 1992} &  
Graphical SFC;\newline  
Free-format ST text edit;\newline  
Block creation;\newline  
Multiple windows;\newline  
Cut and paste between windows;\newline  
Multiple block libraries and projects;\newline  
Simple block print &  
FMC \footnote {Used FMC prototype build facilities, no download or on-line from within GCT} \\ \hline

{April '93} & { Demonstrator} &  
Core Protocol Suite (Resource) based on-line  
communications\footnote{On-line comms depends upon a port of the Resource to Win  
FBD Editor;\newline  
Structured declarations (template) editor;\newline  
Resource Editor\footnote{Includes download for this demonstrator only  
--- download will move to the Network Configuration editor}.  
& PC3000 (LCM demonstrator\footnote{The LCM demonstrator is not a  
feasable version of the PC3000, but an internal  
Milestone});\newline Production Orchestrator \\ \hline

{August '93} & { Demonstrator} & & EPA LIN database  
instruments\footnote{T1000, T100, T640} \\ \hline

{November '93} & {First Release\footnote{Release For Qualification}} & User man  
Professional application documentation\footnote{Application  
documentation may be slipped a release if manpower is not available in time};\ne  
On-line help;\newline  
Instance Browser;\newline  
Network Configuration editor;\newline  
Configuration of operator panel\footnote{Subject to  
approval of ECL Operator Panel project plans} default  
displays\footnote{Including simple alarm management};\newline  
Ladder diagram editor\footnote{The Ladder Diagram editor could be slipped a  
release if needs be};\newline  
PC3000 Hardware editor;\newline  
CDL import (default graphics generation for SFC and FBD)\footnote{For  
CDL import to be useful the existing configuration tools (PS,  
\$mu\$Cell and LINTools) must produce CDL output};\newline  
Tested and defined interface for development of editors by other  
group companies.& PC3000 (Production Version)\\ \hline

March '94 & {Second Release} &  
spreadsheet editor (SFC generation);\newline  
Internationalisation Kit (8-bit character codes);\newline  
Version and release control;\newline  
Mimic configuration for operator panels. & \\ \hline

July '94 & {Third Release} &

Trend and recipe configuration for operator panels;\newline  
 ``Structured`` ST Editor;\newline  
 Simulation.

\end{tabular}

\end{minipage}

\subsection{Metrics}

This section outlines some targets for GCT. The targets are specified in a measurable form, however, the figures are to a large extent an informed guess.

| Attribute   | Scale | First release | Final release | Current products |
|---|-------|---------------|---------------|------------------|
| Tool Portability & Time to move to new computer and/or operating system & 2 elapsed months & 4 elapsed months & 1 year (?)  |       |               |               |                  |
| Application Portability & Time to move a project or parts of a project onto another 1 day & Automatic & Not always possible                                       |       |               |               |                  |
| Application Encapsulation & Time to encapsulate part of an application for re-use 1 day & 1 day & Not always possible   |       |               |               |                  |
| Extensibility & Time to add new editor & x man months & x man months & 6x man months  |       |               |               |                  |
| Modularity & Number of independent parallel development activities & 1 & N (\$>\$1) & 1   |       |               |               |                  |
| Usability & Number of concurrent edit sessions (in separate windows) & N (\$>\$1) & N (\$>\$1) & 1 & Edit / Download cycle time & 2 mins & 0.5 mins & 2 - 20 mins |       |               |               |                  |
| Installability & Who by & End user & End user & End user  |       |               |               |                  |
| Demonstrability & Who by & Engineer & Customer & Sales engineer (?)   |       |               |               |                  |
| Adaptability & Time to add new target & 3 elapsed months & 3 elapsed months & 1 year (if at all)  |       |               |               |                  |
| Large system configuration & Number of concurrent users supported & 1 & N (\$>\$1) & 1  |       |               |               |                  |
| Applications Focus & Who can create focussed packages & N/A & OEM, Systems Division & EI R&D  |       |               |               |                  |

| Attribute   | Scale | First release | Final release | Current products |
|---|-------|---------------|---------------|------------------|
| Qualifications required for use of tool set & By who & Control Eng. & Appl Eng. |       |               |               |                  |

Training & Time to solo & 3 days & 1 day & 3 days \\\hline

ed of Use & Time to solve standard control problems & 2x & x & 2x \\\hline

Likeability & Subjective by poll of users(1 - 10) & 5 & 9 & 7 \\\hline

IEC-1131 conformance & Number of non compliances & Some non-compliances &  
Fully compliant & Base level model is not compliant \\\hline

Software interfaces & Time to interface to a third party config package & 4 man

Testability & Time to produce graphical regression test suite &  
x man months & 2x man months & 6x man months \\\hline

Adherence to graphical user interface standards & User interface  
standards supported & Motif, Windows 3 ( Open Look, Mac \& PM possible) & Motif,  
(Mac, NT, Open Look \& PM possible) & A bit of some standards \\\hline\hline  
\end{tabular}

\newpage

\part{Product Overview}

\section{Outline Of The Product}

The GCT product is designed as a graphical configuration toolset capable of configuring all future, and most current, Eurotherm group control instrumentation. The toolset must be capable of configuring a control system made up of equipment from different Group vendors.

GCT provides an extensible set of editors, allowing the addition of target, industry or application specific configuration methods within a generic framework. The base set of editors includes SFC (Sequential Function Chart), FBD (Function Block Diagram), LD (Ladder Diagram) and ST (Structured Text). Each editor in the base set may also be tuned to match the individual target's feature set. GCT is also reconfigurable itself so that its appearance can be changed to suit the needs of different Eurotherm companies, target markets or products. The provision for industry, application and target specific editors within the toolset ensures that the user view matches his expectations and that he is not forced into thinking in the IEC paradigm.

\epsffig{gct\_arch.eps}{Overview Of GCT Architecture}{GctArch}

As well as adding additional editors, GCT may be subsetting to provide a more limited set of features. For example, a 'Set-up And Monitor' utility may be created by providing only a Network viewer, the Instance Browser and the Template viewer.

GCT editors generate CDL as an intermediate textual language, with the requirement for a product specific 'back-end' to convert the CDL into the run-time format required by the target instrumentation.

Figure~\ref{GctArch} gives an overview of the relationship between the editors, CDL, and back-ends.

The intention is to create a toolset that can configure {\em existing} as well as future products. The only restriction on the suitability of a target is the ability to represent its configuration in CDL. In the long term it is envisaged that {\em new} targets will be designed

which exploit the features of CDL and are hence an ideal match with GCT.

The configuration tool itself is hosted on a computer, with Windows 3 and Unix (Motif) as the initial platforms. Portability to other platforms (particularly Mac, VMS and Windows-NT) will be given consideration but there is currently no intention for the GCT project to be responsible for such ports. Neither is there any intention to provide GCT operation locally on an instrument, although it will be possible to transfer graphical representations of the configuration to the target for local commissioning.

## \section{GCT Application Model}

GCT enables the user to construct control applications based on the IEC 1131 methodology defined in \cite{IECSpec} and developed in \cite{IECJuer}.

This section describes the CDL model as presented by GCT. It is not a requirement that every feature of CDL (and GCT) is supported by a product wishing to use GCT. Section~\ref{targsup} describes how specific targets may use and customise GCT.

## subsection{Application Entities}

Figure \ref{gapehier} illustrates the relationships between the different application entities within this methodology.

\epsffig{gapehier.eps}{Relationship of Application Entities within a Configuration}{gapehier}

### \subsubsection{Top Level Application Entities}

The highest level application entity is known, in IEC terms, as a {\bf Configuration}. Due to the rather overloaded nature of this word the term {\bf Network Configuration} is used in this document. A Network Configuration consists of one or more {\bf Resources} which are networked together. A Resource corresponds to the control strategy running on a largely self-contained physical unit within the Network Configuration, such as a single LCM (Local Controller Module) in a PC3000. In some circumstances it may be appropriate to map a single Resource onto a tightly coupled network of nodes, such as a SSD-Link network.

The Resource level of CDL consists of a set of POUs that can be wired together, and divided into subsets that run under the control of {\bf Task}.

### \subsubsection{Program Organisation Units (POUs)}

Network Configurations, Resources, and Tasks are Blocks that are instanced as they are defined --- there is no concept of a reusable Resource definition for example. POUs are reusable Blocks. That is to

say, they are type definitions that can be instanced as many times as required.

OU may be a {\bf Function Block}, {\bf Function } or a {\bf Program}.

#### \subsubsection{Sub-Block Application Entities}

Within a Block are other application entities that are not stand alone. That is, they only exist within their parent Block and are saved, validated, built, copied, etc., only within the context of their parent.

A Sub-Block may be a {\bf Service }, {\bf Transition}, {\bf Step}, {\bf Action}, {\bf cold start expression}, {\bf wiring expresion}, {\bf variable declaration}, etc..

#### \subsubsection{Application Entity Types}

Supported Application Entity types include:

\begin{itemize}  
\item {\bf Configuration}

Specifies one or more networked resources that are allocated to run on physical es.

\item {\bf Resource}

Contains resource level blocks of type task, program and function block and the wiring between them.

[[{\em A Resource definition should be able to be modified on line. This is desirable to debug parts of an application when other parts of a control system need to be kept running. This concept will be supported by allowing dynamic on-line addition, deletion and rewiring of Resource level POUs. GCT will (in later releases) support this.}]]

\item {\bf Task}

Used to specify the execution rate of blocks that are associated with it. Tasks and blocks associated with tasks are assigned to specific hardware processors or sub-systems.

\item {\bf Program}

A program is a function block that can only be instantiated at the resource level. It usually represents a significant sub-division of a control strategy (e.g. plant unit or machine control). A program can contain action blocks within it.

\item {\bf Function Block}

A function block consists of a number of input parameters that will be processed to generate a number of output parameters. The values of the



input and output parameters will be retained from one execution of a given instance of the block to the next. A function block can contain other function blocks within it.

\item {\bf Service}

A service is a sub-block internal to a program or function block. Services are a CDL extension to the IEC65 standard and are designed to provide a remote procedure call facility and the synchronisation of tasks across the networks. Services provide extra 'methods' for a program or function block resulting in modular block definitions and making it easier to construct distributed applications.

\item {\bf Function}

A function has one or more named input parameters which will be processed to return a value of a specified data type. A function retains no state information from one invocation to the next.

\item {\bf Step}

A sub-block that is internal to a Block that is defined using the SFC graphical language. A step specifies a number of qualified actions that will be executed when the step is active.

\item {\bf Transition}

A sub-block that is internal to a Block that is defined using the SFC graphical language. A transition specifies a condition which will cause previously active step(s) to be cleared and subsequent step(s) to be made active.

\item {\bf Action}

A sub-block that is internal to a Block that is defined using the SFC graphical language. An action specifies some code that will be executed when the steps that it is associated with are active.

\item {\bf Structures \footnotemark[1]}

A sub-block with no associated algorithm, a place holder for sets of related data.

\item {\bf Enumerations \footnotemark[1]}

A user definable set of names representing values that a particular parameter may have. An enumeration defines a new data type.

\item {\bf Variables}

A data item. May be a reference to another data item elsewhere in the Network Configuration.

\end{itemize}

\footnotetext[1]{These are currently not supported by CDL and are included here for completeness; at some time CDL will support them.}

\subsubsection{Application Entity Hierarchy}

\epsffig{blckhier.eps}{Hierarchy Of Application Entities}{blckhier}

As has been said, one of a number of languages or representations may be used to define the body of a POU or sub-block. Many of the languages involve either the explicit or implicit internal declaration of other sub-blocks.

```
\begin{itemize}
\item A configuration may contain resources.
\item A resource may contain task, services, programs and function blocks.
\item A service may contain function blocks and may invoke functions.
A service body may be defined using SFC.
\item A program may contain function blocks and may invoke functions.
A program body may be defined using SFC.
\item A function block may contain function blocks and may invoke functions.
A function block body may be defined using SFC.
\item A block or sub-block with an SFC body (eg service, program,
function block, action), will contain steps, transitions and actions.
\item An action body may be defined using SFC. Alternatively an action may
invoke functions.
\end{itemize}
```

Thus it can be seen that the application model is hierarchical.  
Figure \ref{blckhier}  
shows the hierarchical relationship between the different types of  
Application Entities.

GCT, being based on this hierarchical model, will equally support  
top-down, bottom-up or middle-out configuration of the control strategy.

\subsection{Libraries}

A benefit of the hierarchical, block model is that applications will be  
highly structured and will encourage the reuse of previously defined blocks.  
To this end, GCT incorporates library management features. Blocks  
may be grouped into libraries based on criteria such as specific run-time  
target functionality or an area of application. Applications can make use of  
previously defined libraries. Standard Libraries will be supplied by Eurotherm.  
GCT may also be used to generate user libraries which can in turn be  
used by other applications.

\epsffig{project.eps}{GCT Project}{project}

\subsection{Projects}

A number of network configuration definitions may be used to allocate a set of  
resources to physical nodes on a network in different 'configurations'.  
A number of alternate resource definitions may be used on the same physical  
network to effect different control strategies.  
GCT manages such a set of related network configurations  
and resources in a {\bf project}.

GCT is used to define hierarchical resources, new blocks will be  
created. These blocks will be stored in a library that is private to the  
current project and will not be freely accessible to other applications. It will  
however be possible to use library management features to take a block from a  
project library and include it in a shared library.

Figure \ref{project} Illustrates the entities managed within a GCT project.

## bsection{References}

Resources have the ability to reference blocks and parameters in other Resources, with bindings between Resources being resolved at run-time. This late binding facility is considerably more flexible than the basic IEC model, allowing e.g. a new PO resource to access values from a PC3000 resource which had been previously defined, and without modifying the PC3000 resource.

## \subsection{Copy Protection}

There shall be some optional means of copy protecting GCT.

## \section{GCT Architecture}

Refer to figure-\ref{GctArch} on page-\pageref{GctArch} for an overview of the GCT architecture.

## \subsection{Open Architecture}

### %%\epsffig{gapecflow.eps}{High-Level GCT Dataflow Diagram}{gapecflow}

The GCT software will consist of a suite of front-end utilities that will act on an underlying database (GCT Object Store), supporting creation and management of CDL objects. The GCT architecture will be highly modular, with the disparate utilities having a minimal interface with the rest of GCT (except via the Object Store), such that additional utilities and support for additional target control systems may be easily incorporated as required.

In a later release of GCT it will be possible to incorporate extra utilities without having to relink any GCT released code. Initially, however, all utilities are linked together into one executable file. It will also eventually be possible to supply sub-sets or super-sets of the full GCT editor set so that a range of products may be produced from the one basic set of tools and editors.

Behaviour of GCT, the Object Store, back-ends and the editors can be customised to suit the preferences of an OEM, end user, group company, target product, etc.. Customisation is achieved through the {\bf options database}. This is a set of 'configuration' files (termed database sections) accessed through the object store.

### %%\epsffig{gapetarg.eps}{GCT Dataflow Diagram showing Target Independence}{gapetarg}

## \subsubsection{Use Of CDL}

provides a canonical, portable, extensible representation of all aspects of a control strategy. The use of this common representation will allow the sharing of tools and of application code between multiple configurers and target systems.

GCT will be able to import free format CDL definitions and present

them for display and editing. GCT editors will also support the generation of default graphics to allow a plain textual definition to be viewed graphically.

#### \subsubsection{CDL Object Store}

The purpose of the CDL Object Store is to offload all aspects of managing CDL objects from the various front-end utilities that wish to manipulate CDL objects. Functions such as creation, parsing, storing, loading, browsing, code generation, on-line access, download, etc will be handled by the Object Store in a consistent way and will not have to be reproduced and maintained by multiple applications.

The CDL Object Store will be designed to be a network server that uses CORE protocols for communication with editors\footnote{It will not be implemented as such until a future release of GCT}.

#### \subsubsection{Incorporating Additional Front-End Utilities}

In the GCT architecture, the CDL Object Store will support all the aspects of the creation and management of CDL objects. This means that all editors that are capable of manipulating CDL objects may use the Object Store and will not have to support this management themselves.

An example of this would be the MicroCell spreadsheet representation of an SFC:-

```
\begin{list}{}
\item The spreadsheet representation uses columns to represent
steps. Each column has a transition associated with it and the next
step(s) to make active. Each row has a variable associated with it.
A value (or expression) in a cell directs the assignment of that value
(or the result of the evaluation of the expression) to the row
variable during the column step.
\item A spreadsheet editor may be
incorporated into GCT by calling Object Store transactions to
create steps, transitions and actions. The Object Store would then
handle the parsing, storing and code generation for that CDL
definition.
\item Other information required by the spreadsheet editor may be handled by
the Object Store using attributes. CDL supports the definition of user
attributes that may be associated with any CDL object. For example,
the spreadsheet editor might use attributes to save the row numbers
used by the named variables that will reside in the left hand column
of the spreadsheet.
\end{list}
```

If required, a front-end utility may use the options database to save characterisation parameters.

#### \subsubsection{Incorporating Additional Target Support}

label{targsup}

The Object Store generates CDL for the specification of all aspects of a control strategy. CDL is an extensible language and may be used to include user defined attributes. It is this use of a well defined, canonical representation that enables GCT to be a generic configuration

environment for multiple targets.

GCT may be configured to allow support of a target system provided that the control model for the target system can be represented by CDL (or a subset of CDL). The target need not support all aspects of CDL. E.g., not all targets will support user defined block types. They correspond well to the Resource level of CDL. Examples of such systems are T1000 and (possibly) SSD Link.

GCT is configured for a particular target by setting the appropriate parameters in the options database. These parameters indicate the program, batch-/ script file or in-built utility to use to achieve the target specific functionality.

In order to provide support for a new target the areas to be addressed include:-

```
\begin{itemize}
\item {\bf Target Run-Time Format Generation}
```

The CDL Object Store outputs the definition of a control strategy in CDL text.

A translation utility must be developed to generate target run time-format from the CDL definition. A defined procedure will allow this utility to be installed into the GCT environment and will be invoked by the Object Store for code generation for the given target. The translation utility may use the CDL Object store interface to parse, understand and translate CDL.

Initially the Object Store may also be used to translate CDL code into C that can be compiled and executed in a standard way (as done in the PC3000, PO and FMC).

A group standard for interpretive code that can be generated from CDL should be defined [{\em Future Activity}]. This interpretive code should be generated by the Object Store.

```
\item {\bf Target Load and Initialisation}
```

After the target run-time format has been generated, a final process must be able to take the target code, and do whatever processing is required to download it to the target. GCT will use the Core Protocol Suite FILE and REX (Remote Executive) protocols to perform download and initialisation functions. Targets not supporting these protocols must provide a mapping between them and equivalent mechanism for the target.

```
\item {\bf Target Comms Driver}
```

The Object Store will provide a generic interface for on-line access from GCT utilities to live target systems. GCT will use the Core Protocol Suite RDP (Realtime Database Protocol) to perform on-line access.

```
\item {\bf Target Specific Front-End Utility}
```

A target specific front-end utility may be required. This can be incorporated

into GCT as for any other front-end utility.

item{\bf Customisation Of Generic Tools}

Generic editors may be customised according to the capabilities of the target product --- for example to limit the size of an SFC (number of steps, or number of actions per step). This customisation is held in the options database and is accessed via the object store.

\end{itemize}

\subsection{Multiple Platforms}

GCT will be portable across the X window system (initially on Interactive 386/ix) and Microsoft Windows 3 (version 3.1 upwards).

The OSF/Motif library will be used in the construction of the version of GCT for X, and this version will be designed to work best when used in conjunction with the Motif Window Manager.

On PCs, GCT will support the following display standards: EGA, VGA, Super VGA (resolution of 800  $\times$  600 upwards recommended but not necessary). Due to the device independent nature of X and MS Windows, support for future display standards of higher resolution (e.g. XGA) will be automatic. The minimum PC platform for GCT will be a 20MHz 386 with 4Mb RAM, 40Mb hard disk.

\subsection{Construction}

The GCT development will use XVT, a high-level GUI toolkit, to increase productivity and ensure consistency across supported platforms. Use of XVT would also allow portability to Mac and OS/2. GCT will also support interfacing with other third party graphics libraries (e.g. GMS on Unix).

GCT will be coded primarily in C++, and will reuse code (either C or C++) from existing programming tools where possible.

\subsection{Operating environment}

GCT will use the Core Protocol Suite (Realtime Database Protocol, File Protocol and Remote Execution Protocol) to communicate with targets. In this way GCT will fit into a network of Core compliant products.

The network version of the CDL Object Store will use the Core Communication Messaging Services to provide Object Store services, hence any node on a Core network should be able to query the object store for information.

\section{General User Interface Features}

The GCT user interface will adopt the native 'look and feel' of the operating system/.graphics environment under which it runs (e.g., using the multiple document interface of Windows 3). This means that GCT appearance will vary slightly according to the host platform.

At the highest level GCT will be driven from a horizontal menu bar

which will remain at the top of the screen (Windows 3) or the top of the main window (Motif).

GCT utilities will have a common 'look and feel'. Utilities will run inside windows and use will be made of dialog boxes, list boxes, icons, and other standard widget types, to facilitate user interaction.

General user interface requirements may be summarised:

```
\begin{description}
\item[Consistent:] Use of user interface constructs for similar operations will
\item[Compliant:] The design and behaviour of the user interface must be in
accordance with accepted GUI standards such as Motif and Windows style
guidelines.
\item[Responsive:] The user interface must appear responsive to user action at
all times.
\item[Input devices:] GCT will be driven most efficiently using a
mouse, but it will be possible to work with keyboard only.
\item[Accelerators:] Accelerator keys will be defined for rapid access to
frequently used menu options.
\item[Non-modal:] GCT will contain minimal modality; in general, all utilities w
\item[Multi-window:] The user will be able to view and modify multiple areas of
the configuration simultaneously, using different utilities or multiple copies
of the same utility as required.
\item[Interoperable:] It will be possible to exchange data between GCT
utilities, or between a GCT utility and third party utilities, using a
clipboard and other mechanisms (such as DDE) where appropriate.
\item[Multi-user:] It will be possible for multiple users to work on
applications simultaneously across a network; library management will control
locking and updating of the various components of a configuration by
employing the version control facilities.
\item[International:] GCT is targeted at an international market and
will therefore provide full multi-language support (8-bit character
sets), also taking into account date and time formats etc.
\end{description}
```

#### ``` \subsection{Introduction To Menu Bar Usage} ```

GCT menu bar functions can be divided into three categories:-

```
\begin{description}
\item[{\bf Main Menu Functions}] which
allow access to operations affecting the entire GCT environment.
\item[{\bf Block Menu Functions}]
which contain commands affecting a block (such as save to file or
build executable).
\item[{\bf Editor Menu Functions}]
which provide editor specific functions (such as deleting a marked
section of ST text or placing a transition in an SFC).
\end{description}
```

Under Windows 3 the menu bar at the top of the screen changes according to the current active window. Under Motif each window has its own menu bar according to its functionality.

The following description of techniques for accessing menu bar commands applies equally to all menu bars.

The structure of a menu bar consists of a horizontal menu bar, whose

entries corresponds to a pull-down menu of commands related to a particular area of functionality (e.g. file handling or accessing help).

Items in the menu bar may be accessed using the mouse by clicking on the required item, or using the keyboard by pressing the Alt key in conjunction with the underlined (usually the first) letter of the item (e.g. Alt-F to access the File menu).

Having selected a particular pull-down menu, items within the pull-down may be selected either by clicking on the item with the mouse, or by pressing the letter of the item which is underlined (e.g. S for Save or X for Exit, both of these being in the File pull-down).

Some regularly used commands from the pull-down menus will have corresponding accelerator keystrokes, so that a single keystroke can activate the command without first having to access a pull-down menu. Where a menu item has a corresponding accelerator key sequence, this sequence will appear alongside (to the right of) the item name in the pull-down menu. At present, this specification does not address the assignment of specific accelerators, although a number of these will be determined by style guidelines.

Menu items which lead to a dialog box (e.g. to allow the setup of command options, or for confirmation) are followed by an ellipsis (...) when displayed.

```
newpage
part{Editors and Utilities}
```

```
\section{GCT Views}
\label{viewsgct}
```

```
\subsection{Philosophy of Operation}
```

```
\subsubsection{View Types}
```

There are two types of GCT views. The first type are those that are displayed in the main GCT window, and that are controlled by the main menu bar --- these are termed {\em main views}. The second type are displayed in their own windows with their own menu bar these are termed {\em independent views}. There is only ever one main view active at a time, but there can be multiple active independent views. Note that under Windows 3 there is only one menu bar. Its contents change according to the currently active window.

Sections \ref{MainViews} and \ref{IndepViews} set out the primary views in GCT, and the operations supported by each view. The specific details of the user interaction for each operation are not included but an overview of the style of interaction is given as an introduction.

\*Diagram of views

```
\sffig{views2.eps}{Relationships between GCT Views}{views2}
```

Figure~\ref{views2} (page~\pageref{views2}) gives an overview of the relationship between the different views within GCT. A labelled box in the diagram represents a GCT view. The boxes are labelled with the section number in this document that describes that view. An arrow



indicates that another view may be opened from that view.

#### \subsubsection{Object Types}

GCT operates on the following types of object,

\begin{description}  
\item{The Project}

GCT operates within a project or library context. The project acts as a 'folder' within which Resources, Configurations and the Project Library are stored. The project context maps directly onto a directory.

A Project is represented and managed by the Project main view (see section~\ref{ProjectView} and figure~\ref{project\_view}, page~\pageref{project\_v

\item{Resources}

The Resource represents the control strategy executing within a node or tightly coupled network.

Resources belonging to a project are listed and managed in the Project main view.

A Resource is represented by a Block independent view by the editor that was used to create the Resource (see section~\ref{BlockView}).

\item{Configurations}

Configurations (called ''Network Configurations'' in the tool) represent the distribution of Resources across physical networks. Configurations belonging to a project are accessed and managed via the Project main view.

A Configuration is represented by the Network Configuration main view (see sect

\item{Project Library}

The Project Library is a folder containing all user defined block types that are created within a project. Its behaviour and facilities are identical to those of a Shared Library. The Project Library is accessed and managed via the Open Project main view (see section~\ref{ProjectView}) or a Library Browser independent view (see section~\ref{LibraryBrowse}).

\item{Shared Libraries}

A Shared Library is a folder containing user defined block types that may be used by more than one project. Their behaviour and facilities are identical to those of a Project Library. Shared Libraries are accessed and managed via the Open Library main view (see section~\ref{ProjectView}) or a Library Browser independent view (see section~\ref{LibraryBrowse}).

\item{Block Types}

Block types are created in a Resource or Library context. They are always saved in a library (Project or Shared). Block types are managed in the Open Project~/ Library main view (see section-\ref{ProjectView}) or a Library Browser independent view (see section-\ref{LibraryBrowse}).

A block type is represented in a Block independent view by the editor that was used to create it.

\end{description}

\subsubsection{Main Menu}

The menu for the main window operates on items in the current main view and also provides other functions global to the GCT environment. The menu bar window will, by default, remain permanently at the top of the screen (Windows 3) or top of the main window (Motif) while GCT is running.

The user will be able to select options from the main menu at any time and with any combination of views on screen, inappropriate menu items being 'greyed out' to indicate their unavailability.

The layout of the items in the menu bar will be based on the following outline:-

```
{\samepage
\begin{tabular}{|l|lllp{9cm}r|}
\verb+File+ & \verb+(View)+ & \verb+Utilities+ & \verb+Window+ & \verb+Help+ \\
\hline
\end{tabular}
\begin{tabular}{|l|p{5in}}
\cline{1-1}
\verb+Project/Library Manager+ & Changes main view to the project~/
library management view (\ref{prolib_view})\\ \cline{1-1}
\verb+Close Project+ & Closes the current project [library]. Greyed
out if no project [library] open.\\
\verb+Project View+ & Changes the main view to the Open Project~/
Library view (\ref{ProjectView}). Greyed
out if no project [library] open. \\ \cline{1-1}
\verb+New...+ & Creates a new POU. Greyed
out if no project [library] open.\\
\verb+Open...+ & Opens a POU.\\ \cline{1-1}
\verb+Print Setup...+ & {\em Windows 3 only} Enters printer set up dialog. \\ \c
\verb+Exit+ & Exits the tool \\ \cline{1-1}
\end{tabular}
```

` % end samepage

```
{\samepage
\begin{tabular}{|l|l|lllp{9cm}r|}
\hline
\verb+File+ & \verb+(View)+ & \verb+Utilities+ & \verb+Window+ & \verb+Help+
```

```

\hline
\end{tabular}
\begin{tabular}{p{0.3in}|l|p{4.7in}} \cline{2-2}
    tem 1 & View specific menu items\\ \cline{2-2}
& etc. & \\ \cline{2-2}
\end{tabular}

} % end samepage

```

For example, the \verb"(View)" menu may be used by the network editor to provide an \verb"On Line" menu.

```

{\samepage
\begin{tabular}{|l|l|p{9cm}r|} \hline
\verb+File+ & \verb+(View)+& \verb+Utilities+ & \verb+Window+&& \verb+Help+ \\ \hline
\end{tabular}
\begin{tabular}{p{0.9in}|l|p{4.1in}} \cline{2-2}
& \verb+Browse...+ & Invoke the instance browser \\
& \verb+Preferences...+ & Define user preferences\\
& \verb+Setup...+ & GCT Setup sub-menu \\
& etc. & Other registered utilities\\ \cline{2-2}
\end{tabular}

} % end samepage

```

```

{\samepage
    following on {\em Windows 3 only}

\begin{tabular}{|l|l|p{9cm}r|} \hline
\verb+File+ & \verb+(View)+& \verb+Utilities+ & \verb+Window+&& \verb+Help+ \\ \hline
\end{tabular}
\begin{tabular}{p{1.725in}|l|p{3.2in}} \cline{2-2}
& \verb+Tile+ & Arrange all windows as tiles \\
& \verb+Cascade+ & Cascade all windows \\
& \verb+Arrange Icons+ & Arrange iconised windows tidily\\ \cline{2-2}
& $\surd$ \verb+(window 1)+ & Currently selected window\\ \cline{2-2}
& etc. & other windows \\ \cline{2-2}
\end{tabular}

} % end samepage

```

#### \subsubsection{Block Menu}

The menu on a block edit window contains items relating to the block (e.g. Save, Validate) and items relating to the block's editor.

The common items to all block menus are as follows

```

{\samepage
    f Motif :-}

\begin{tabular}{|l|l|p{9cm}r|} \hline
\verb+File+ & \verb+Edit+& \verb+(Editor 1)+ & \verb+(Editor 2)+ && \verb+Help+ \\ \hline
\end{tabular}

```

```

\begin{tabular}{|l|p{5in}} \cline{1-1}
\verb+Close+ & Close the block edit (prompts if there are unsaved changes)\
\verb+Save+ & Save block definition to current project~/ library \
\verb+Save As...+ & Save block definition to current project~/ library
under a new name \
\verb+Validate+ & Validate block definition \
\verb+Build+ & Create the
target representation of the block\
\verb+Sub-Blocks...+ & Invoke sub-block management dialog\ \cline{1-1}
\verb+Print+ & Print the block definition\ \cline{1-1}
\end{tabular}

} % end samepage

```

```

{\samepage
\begin{tabular}{|l|l|lp{9cm}r|} \hline
\verb+File+ & \verb+Edit+ & \verb+(Editor 1)+ & \verb+(Editor 2)+ & \verb+Help-
\hline
\end{tabular}
\begin{tabular}{p{0.29in}|l|p{4.7in}} \cline{2-2}
& \verb+Undo+ & Undo the last change \ \cline{2-2}
& \verb+Cut+ & Cut the selected items into the clipboard\
& \verb+Copy+ & Copy the selected items into the clipboard\
& \verb+Paste+ & Paste the clipboard items into the window\ \cline{2-2}
& \verb+Delete+ & Deletes the selected items without copying to the clipboard\
& \verb+Undo All+ & Undo all changes since last save \ \cline{2-2}
\end{tabular}

} % end samepage

```

```

{\samepage
\begin{tabular}{|ll|l|lp{9cm}r|} \hline
\verb+File+ & \verb+Edit+ & \verb+(Editor 1)+ & \verb+(Editor 2)+ & \verb+Help-
\hline
\end{tabular}
\begin{tabular}{p{0.75in}|l|p{4.3in}} \cline{2-2}
& \verb+Item 1+ & Editor specific menu items \
& \verb+etc.+ & \ \cline{2-2}
\end{tabular}

} % end samepage

```

```

{\samepage
\begin{tabular}{|lll|l|lp{9cm}r|} \hline
\verb+File+ & \verb+Edit+ & \verb+(Editor 1)+ & \verb+(Editor 2)+ & \verb+Help-
\hline
\end{tabular}
\begin{tabular}{p{1.63in}|l|p{3.4in}} \cline{2-2}
\verb+Item 1+ & Further editor specific menu items \
\verb+etc.+ & \ \cline{2-2}
\end{tabular}

} % end samepage

```

```
{\bf Windows 3 :-}
```

On Windows 3, since there is only one menu at the top of the screen,  
Block menu also holds some of the Main menu items.

```
{\samepage
\begin{tabular}{|l|l|lllp{7.5cm}r|} \hline
\verb+File+ & \verb+Edit+ & \verb+(Editor 1)+ & & \verb+(Editor 2)+
& \verb+Window+ & & \verb+Help+ \\\hline
\end{tabular}
\begin{tabular}{|l|p{5in}} \cline{1-1}
\verb+Project/Library Manager+ & Changes main view to the project~/
library management view (\ref{prolib_view})\\ \cline{1-1}
\verb+Close Project+ & Closes the current project [library].\\
\verb+Project View+ & Changes the main view to the Open Project~/
Library view (\ref{ProjectView}). \\\cline{1-1}
\verb+New...+ & Creates a new POU.\\
\verb+Open...+ & Opens a POU.\\ \cline{1-1}
\verb+Close+ & Close the block edit (prompts if there are unsaved changes)\\
\verb+Save+ & Save block definition to current project~/ library \\\
\verb+Save As...+ & Save block definition to current project~/ library
under a new name \\\
\verb+Validate+ & Validate block definition \\\
\verb+Build+ & Build block for target execution\\
\verb+Sub-Blocks...+ & Invoke sub-block management dialog\\ \cline{1-1}
\verb+Print+ & Print the block definition\\
\verb+Print Setup...+ & Enters printer set up dialog. \\\cline{1-1}
\verb+Exit+ & Exits the tool \\\
\verb+About...+ & Shows the GCT 'About' text. \\\cline{1-1}
\end{tabular}

} % end samepage
```

```
{\samepage
\begin{tabular}{|l|l|lllp{7.5cm}r|} \hline
\verb+File+ & \verb+Edit+ & \verb+(Editor 1)+ & & \verb+(Editor 2)+
& \verb+Window+ & & \verb+Help+ \\\hline
\end{tabular}
\begin{tabular}{|p{0.29in}|l|p{4.7in}} \cline{2-2}
& \verb+Undo+ & Undo the last change \\\cline{2-2}
& \verb+Cut+ & Cut the selected items into the clipboard\\
& \verb+Copy+ & Copy the selected items into the clipboard\\
& \verb+Paste+ & Paste the clipboard items into the window\\ \cline{2-2}
& \verb+Delete+ & Deletes the selected items without copying to the clipboard\\
& \verb+Undo All+ & Undo all changes since last save \\\cline{2-2}
\end{tabular}

} % end samepage
```

```
\samepage
\begin{tabular}{|l|l|lllp{7.5cm}r|} \hline
\verb+File+ & \verb+Edit+ & \verb+(Editor 1)+ & & \verb+(Editor 2)+
& \verb+Window+ & & \verb+Help+ \\\hline
\end{tabular}
```

```

\begin{tabular}{p{0.75in}|l|p{4.3in}} \cline{2-2}
& \verb+Item 1+ & Editor specific menu items \\
& \verb+etc.+ & \\ \cline{2-2}
& \end{tabular}

} % end samepage

```

```

{\samepage
\begin{tabular}{|l|l|p{7.5cm}r|} \hline
\verb+File+ & \verb+Edit+ & \verb+(Editor 1)+ & \verb+(Editor 2)+
& \verb+Window+ & & \verb+Help+ \\ \hline
\end{tabular}
\begin{tabular}{p{1.632in}|l|p{3.4in}} \cline{2-2}
& \verb+Item 1+ & Further editor specific menu items \\
& \verb+etc.+ & \\ \cline{2-2}
& \end{tabular}

} % end samepage

```

```

{\samepage
\begin{tabular}{|l|l|l|p{7.5cm}r|} \hline
\verb+File+ & \verb+Edit+ & \verb+(Editor 1)+ & \verb+(Editor 2)+
& \verb+Window+ & & \verb+Help+ \\ \hline
\end{tabular}
\begin{tabular}{p{2.547in}|l|p{2.4in}} \cline{2-2}
& \verb+Tile+ & Arrange all windows as tiles \\
& \verb+Cascade+ & Cascade all windows \\
& \verb+Arrange Icons+ & Arrange iconised windows tidily \\
& \verb+($\verb+(window 1)+ & Currently selected window \\
& etc. & other windows \\ \cline{2-2}
& \end{tabular}

} % end samepage

```

\subsubsection{Sub-Block Menu}

When a sub-block is opened from within a block edit a restricted block menu is available. Menu items which relate to the whole block are not available on the sub-block. Only the first pull-down menu is affected :-

```

{\samepage
{\bf Motif :-}

\begin{tabular}{|l|l|l|p{7.5cm}r|} \hline
& \verb+Sub-Block+ & \verb+Edit+ & \verb+(Editor 1)+ & \verb+(Editor 2)+ & & \verb+etc.+ \\ \hline
\end{tabular}
\begin{tabular}{|l|p{5in}} \cline{1-1}
\verb+Close+ & Close the sub-block edit \\
\verb+Validate+ & Validate sub-block definition \\ \cline{1-1}
\end{tabular}

```

```

\verb+Sub-Blocks...+ & Invoke sub-block management dialog. Greyed out
unless this sub-block has sub-blocks of its own.\\ \cline{1-1}
\verb+Print+ & Print the sub-block definition\\ \cline{1-1}
    id{tabular}

```

```

} % end samepage

```

```

{\samepage
{\bf Windows 3 :-}

```

```

\begin{tabular}{|l|l|l|l|p{7.5cm}|r|} \hline
\verb+File+ & \verb+Edit+ & \verb+(Editor 1)+ & & \verb+(Editor 2)+ & \\
& \verb+Window+ & & & \verb+Help+ & \\
\end{tabular} \hline
\begin{tabular}{|l|p{5in}|} \cline{1-1}
\verb+Project/Library Manager+ & Changes main view to the project~/
library management view (\ref{prolib_view})\\ \cline{1-1}
\verb+Close Project+ & Closes the current project [library].\\
\verb+Project View+ & Changes the main view to the Open Project~/
Library view (\ref{ProjectView}). \\ \cline{1-1}
\verb+New...+ & Creates a new POU.\\
\verb+Open...+ & Opens a POU.\\ \cline{1-1}
\verb+Close+ & Close the sub-block edit\\
\verb+Validate+ & Validate sub-block definition \\ \cline{1-1}
\verb+Sub-Blocks...+ & Invoke sub-block management dialog. Greyed out
unless this sub-block has sub-blocks of its own.\\ \cline{1-1}
\verb+Print+ & Print the sub-block definition\\
    \verb+Print Setup...+ & Enters printer set up dialog. \\ \cline{1-1}
\verb+Exit+ & Exits the tool \\ \cline{1-1}
\end{tabular}

```

```

} % end samepage

```

```

\subsubsection{Help Menu}

```

The help menu is common to all menu bars  
This menu will give a number of access points to the help system.

The Help pull-down menu contains the following items:

```

\begin{center}
\begin{tabular}{|l|l|l|l|} \cline{1-1}
\verb+Index+ & Opens a help system index window & & \\
\verb+Menu Bar+ & Help on the menu bar & & \\
\verb+Keyboard+ & Help on using a keyboard with GCT & & \\
\verb+Mouse+ & Help on using a mouse with GCT & & \\
\verb+Tutorial+ & GCT tutorial? & & \\
& \cline{1-1}
\verb+About+ & GCT copyright and version information & & \\
& \cline{1-1}
\end{tabular}
\end{center}

```

```

\subsection{Main Views}
\label{MainViews}

```

Main views all occupy the main window. They are controlled from the main menu bar.

\subsubsection{Initial View}

\label{prolib\_view}

The initial view presented on start up has the GCT welcome message and the project~/ library management view.

The welcome message gives version and copyright information.

The project~/ library management view lets the user

\begin{itemize}

\item Open / Create a project

\item Open / Create a library

\item Perform other management functions such as deleting and renaming a project or library.

\end{itemize}

\subsubsection{Open Project~/ Library View}

\label{ProjectView}

%Diagram of views

\epsffig{projectv.eps}{Main Project View}{project\_view}

Figure-\ref{project\_view} illustrates the Open Project~/ Library view. Projects and libraries behave in very similar ways, the main difference being that a library cannot contain either Configurations or Resources. Project and library block management functions are accessed as ``Other Operations''. These include \verb"Copy, Move, Rename" and \verb>Delete".

The view lists all the blocks (Function Blocks, Functions, and Programs, together with Resources and Configurations if appropriate) that are in the project or library. Blocks can be opened for edit and new blocks can be created.

Blocks selected from the scroll list can be copied, moved, deleted, printed, renamed, etc. via the ``Other Operations'' button. The library search path (other libraries which are searched for block types not contained in the currently opened project or library) is managed via the ``libraries'' button.

\subsubsection{Network Configuration View}

\label{configuration\_view}

The network configuration view associates Resources with target nodes. Functionality includes

\begin{itemize}

\item Specifying which Resources are to run on which nodes.

\item Defining network characteristics



```

\item Downloading, starting, stopping Resources
\item Launching a Resource editor in on-line mode.
\item Network monitoring and debugging
\item Performing inter-Resource wiring functions (Var References).
\end{itemize}

```

```

\subsubsection{Target Support}

```

Utilities can be incorporated to allow configuration of target specific aspects of any target system.

```

\subsection{Independent Views}
\label{IndepViews}

```

Independent views exist in windows separate from, and largely independent of, the main view. There can be multiple active independent views, most of which will have their own menu bars. Under Motif each independent view has its menu in its own window. Under Windows 3 the current active window shows its menu bar at the top of the screen.

```

\subsubsection{Block View}
\label{BlockView}

```

The GCT block view will allow the display and modification of all application block types.

The block editor will consist of a one or two pane window and a menu bar, with the panes separated by a sash. The two panes will contain the block's declarations in one pane and the body in the other. [There is a current discussion as to whether declarations and body should be in separate windows, not separate panes of the same window. Panes may be retained to show debug data, etc.]]

The declarations are the interface to the block and the internals of the block (see section \ref{DeclarationsEditors} for details of the Declarations Views - the different views presented in the declarations pane). The body is the algorithm of the block (see sections \ref{STEditor} to \ref{FBDEditor} for details of the Body Editors - the different views presented in the body pane).

```

{\bf Declarations}

```

A block declarations consist of its interface and its internals.

The interface of a block is the parameters that may be viewed and used by other blocks (i.e. its inputs and outputs). Interface parameters are also accessible from within the implementation of the body of the

block.

The internals of a block are the parameters and other CDL items that can be referenced {\em only} from within the implementation of the body of the block, they cannot be seen from outside. Some internals are implicitly declared, e.g. an SFC body includes internals which are Steps, Transitions and Actions.

The representation of the interface and internals declarations is common to all block types and is independent from the implementation of the body.

The declarations pane provides a generic representation of all application block types.

The interface editor will support:

```
\begin{itemize}
\item Display interface and internal declarations graphically or textually.
\item Declare interface and internal variables and attributes
\item Interactive help.
\end{itemize}
```

{\bf Body}

The body of a block is the actual algorithm that is associated with the block. The body may be implemented using one of a suite of IEC languages (e.g. SFC, ST, etc), or using another representation (e.g. C, spreadsheet, etc).

The representation of the body of a block will depend on how it was originally defined. When the body of a new block is first opened for edit, an option is given to select one of a number of editors that are appropriate for the given block type. The selected editor will then be invoked to allow the body to be defined. Thereafter the body will retain its body editor type and whenever it is selected for display or modification, the respective editor will again be invoked.

In some cases it will be possible to display or transform a block body into a simpler representation (usually to ST text), eg SFC to ST. It will not in general be possible to transform to a more complicated body type (although default generation of SFC and FBD graphics from ST text will be supported).

{\bf Block and Sub-Block Types}

The following sub-sections describe features of the specific types supported. In each case editors that are appropriate for the definition of a block of the given type are listed. (Other editors may be appropriate e.g Spreadsheet for defining any SFC.)

```
\begin{itemize}
\item Configuration:
```

To be defined (the first release of GCT will not include the configuration level in other words, configurations will consist of a single resource)

```

\begin{itemize}
\item Interface: None.
\item Internals: Resource declarations.
\end{itemize}

\item Resource:

\begin{itemize}
\item Interface: None.
\item Internals: Task, Service, Program, Function Blocks
\item Body may be edited by: ST, FBD.
\end{itemize}

\item Task:

\begin{itemize}
\item Body : A list of POU-task associations may be displayed.
\end{itemize}

\item Service:

\begin{itemize}
\item Interface: Inputs, Outputs.
\item Internals: declarations (includes steps, transitions and actions if SFC
body).
\item Declarations may be edited by: ST, FBD, Tempalte Editor
\item Body may be edited by: ST, LD, SFC, FBD.
\end{itemize}

\item Program:

\begin{itemize}
\item Interface: Inputs, Outputs
\item Internals: declarations (includes steps, transitions and actions if SFC
body).
\item Declarations may be edited by: ST, FBD, Tempalte Editor
\item Body may be edited by: ST, SFC, LD, FBD.
\end{itemize}

\item Function Block:

\begin{itemize}
\item Interface: Inputs, Outputs, Input\_Outputs.
\item Internals: declarations (includes steps, transitions and actions if SFC
body).
\item Declarations may be edited by: ST, FBD, Tempalte Editor
\item Body may be edited by: ST, SFC, LD, FBD.
\end{itemize}

\item Function:

\begin{itemize}
\item Interface: Inputs, implicit typed output
\item Internals: declarations.
\item Declarations may be edited by: ST, FBD, Tempalte Editor
\item Body may be edited by: ST, LD, FBD.
\end{itemize}

\item Step:

```

```

\begin{itemize}
\item Interface: Executing, Time, (Finished) parameters.
    em List of actions associations and qualifiers may be edited
\end{itemize}

```

\item Transition:

```

\begin{itemize}
\item Interface: None.
\item Internals: None.
\item Body may be edited by: ST, LD, FBD.
\end{itemize}

```

\item Action:

```

\begin{itemize}
\item Interface: Output (need clarification from IEC standard).
\item Internals: None - uses declarations of POU in which action is defined.
Will include steps and transitions if SFC body.
\item Body may be edited by: ST, SFC, LD, FBD.
\end{itemize}

\end{itemize}

```

```

\subsubsection{Library Browser}
\label{LibraryBrowse}

```

.. \em The library browser been absorbed into the Open Project~/  
Library view (\ref{ProjectView}})]

```

\subsubsection{Instance Browser}
\label{InstanceBrowse}

```

The Instance Browser [{\em or Hierarchy Browser}] is used to peruse a hierarchy instances. This hierarchy may be rooted from either a type definition (exploring blocks instanced within the type) or from a Resource or Network editor.

The browser can be used,

```

\begin{itemize}
\item To explore the block hierarchy.
\item To select (one or more) objects from the block hierarchy,
    pasting their full hierarchic names on a clipboard
\item To 'open' an object from the block hierarchy. Opening a block
    displays the template view of the object (if the hierarchy root is
    on-line the template view is in On-Line monitor mode; else it is in a
    non-edit mode). From the template view it should be possible to
    launch an edit of the object's type definition.
\item Do queries, for example find all the
    blocks with \verb"TEST_ENABLE == TRUE"
\end{itemize}

```

The browser should support the following features,

```

\begin{itemize}
\item Selection of an instance by name completion
\item Non-hierarchic searching for a named instance

```

```

\item Filtering blocks by type or partial name (including wild cards)
\item Selection of an instance name which is transferred to the clipboard.
\end{itemize}

```

## ``` \subsubsection{Trace} ```

A trace facility will provide a non-intrusive means of run-time diagnostics. A trace specification may be defined and downloaded and trace data can be collected from the target system and displayed. Trace specifications can be stored on file for later retrieval.

[[\em Trace is only valid when supported in the target. Isn't it part of type debug mode of any editor rather than a separate view? Is it a target facility? or a general target facility?]]

```

\section{Declarations Views}
\label{DeclarationsEditors}

```

The Declarations Views are used in four modes,

```

\begin{description}
\item{\bf Type Edit:} During { block edit} a declaration view is used
in the declarations pane to edit the definition of the interface and
internals of the block.

\item{\bf Instance Configuration:} During { block edit}, a declaration
view is used to configure the instance of any sub-block. For example,
    set cold start values or attributes such as ``units'' on a block
placed within another block. In this mode only the block's interface
is made visible.

\item{\bf On-Line Monitor:} When a block (instance) is viewed on-line,
a declaration view is used to display and interact with the live
values of the block's interface variables. In this mode only the
block's interface is made visible.

\item{\bf Type Debug:} When debugging a new block type, the user
wishes to use the same editor as used to create the block operating in
an ``on-line view mode'' as a source level debugger. Each declarations
view must therefore be capable of on-line operation to support type
debugging.

```

```
\end{description}
```

GCT will provide three different declarations views,

```

\begin{description}
\item{\bf ST:} The ST declarations view is a raw text editor (see
section \ref{STEditor} for further details). The declarations are
entered as ST text (e.g. \verb"VAR_INPUT x:DINT; END_VAR").

\item{\bf Template Editor:} The template editor lists parameters
according to their mode (inputs, outputs, internals) and provides a
menu based mechanism for their definition~/ modification (see section
\ref{TemplateEditor} for further details).

\item{\bf FBD:} The FBD editor provides a purely graphical view of a

```

block's declarations and the wiring between them (see section \ref{FBDEditor} for further details). Note that using FBD on its own generates an automatic body that invokes all of the blocks in order to effect the wiring. Use of FBD declarations with any other form of body is allowed, but there is no automatic invocation of the blocks --- it is up to the user to specify the required invocation in the body method he chooses.

\end{description}

Not all declarations views can be used in every mode:-

|                 | ST  | FBD | Template | On-Line Monitor | Type Debug |
|-----------------|-----|-----|----------|-----------------|------------|
| ST              | Yes | Yes | Yes      | Yes             | Yes        |
| FBD             | Yes | Yes | Yes      | Yes             | Yes        |
| Template        | Yes | Yes | Yes      | Yes             | Yes        |
| On-Line Monitor | Yes | Yes | Yes      | Yes             | Yes        |
| Type Debug      | Yes | Yes | Yes      | Yes             | Yes        |

\end{tabular}

Notes,

\begin{enumerate}

\item Although the ST or FBD editors could be used in On-Line Monitor mode it doesn't make much sense. After all if you want access to the full type definition you're in Type Debug mode --- otherwise you only have access to the interface and the template editor is the standard interface representation.

\item In type debug mode the ST editor requires some dialog capability. E.g., selecting a piece of text will bring up a list of all the variables named in that text together with their live values. Selecting any of these variables will bring up an interaction dialog which can be used to change the value [{\em possibly in a bottom pane?}].

\item Any changes to values in Type Debug mode can be captured and selectively applied to the type definition in type edit mode.

\item The Template Editor in On-Line Monitor mode restricts access to reading and writing values only. In particular, writing to attributes may not be supported by the target. Writing to display attributes that affect the template editor view itself will take immediate effect. Any changes to values in On-Line Monitor mode can be captured and applied to the instance definition in Instance Configuration mode.

\item The FBD Editor can be used for Type Debug : selecting a terminal will pin up the live value at that terminal [{\em possibly in a bottom pane?}]; selecting a block will bring up an On-Line Monitor for that block.

\end{enumerate}

\subsection{Choice Of Editor}

[{\em this section is open for review. It may be that opening a new block just brings up the declarations editor and choice of body editor is

made the first time the body is opened from the declarations editor}}]

The choice of declaration view in type edit mode is made by the associated body editor. The editor (when registering itself to the object store) specifies the default declaration editor and lists the other permissible declaration editors. The user can change the actual declaration view to any of the permissible editors by menu option, and by saving preferences in the options database.

```
\subsection{VARs, VAR REFERENCES, Attributes and I/O}  
\label{VarRef}
```

A VAR REFERENCE is a declaration that is a reference to a (set of) variable(s) outside of the current scope. There should be two ways of declaring VAR REFERENCES,

```
\begin{itemize}  
\item Create the variable as if it were local, then convert it to a  
reference. To support this method the variable declaration dialog  
(Template Editor and FBD Editor) must have an ``is a reference''  
button and a control for configuring the reference (reference string,  
scan rate, etc). The reference string for the variable can be set by  
either direct entry or  
selecting the target object via the instance browser.  
\item Select the ``to be referred to'' variable(s) via the instance  
browser. This should automatically create a VAR REFERENCE of the right  
type. If multiple variables are selected a new type definition should  
be automatically created and instanced.  
\end{itemize}
```

There is a requirement to defer the specification of certain nested instance parameters. For example, the specification of I/O addresses and VAR REFERENCE reference strings may be left as late as Resource configuration, no matter how deeply nested the block. [{\em It is suggested that}] there should be a class of cold start value called a ``configuration input'' (specified by a pre-defined attribute) with the following specific behaviour,

```
\begin{itemize}  
\item They do not appear as normal wireable inputs.  
\item They can only be set from the declaration view Instance  
Configuration mode or in raw ST.  
\item If they are left unspecified in a block type definition they are  
automatically made configuration inputs at the next level out. The  
user can rename them from the declaration editor.  
\end{itemize}
```

```
\subsection{Template Editor}  
\label{TemplateEditor}
```

A template editor is a declaration editor which supports Type Edit, Instance Configuration, On-Line Monitor and Type Debug modes.

[{\em Note that in the following sections the semantics of short and long list parameters are not yet defined}]

\subsubsection{Type Edit Mode}

```
\begin{itemize}
  \item Creation, deletion, modification and renaming of simple and
  compound [\em FB or Structure] variables and constants.
  \item Creation, deletion, modification and renaming of attributes for
  variables.
  \item Specification of cold start expressions for variables
  \item Specification of parameters as short-/ long list with respect to
  listing pins during instance configuration (short list parameters appear by
  default in a wiring diagram, long list require additional action to be
  made visible)
  \item Specification of parameters as RETAIN'd variables.
  \item A template view that shows the block as it will appear when
  instanced within another FBD
\end{itemize}
```

\subsubsection{Instance Configuration Mode}

```
\begin{itemize}
  \item Interface parameters only, defaults to short list
  \item Assignment of cold start expressions
  \item Assignment of attribute values.
  \item Assignment of configuration inputs
  \item Change from short list to long list
  \item Invocation of block type editor.
  \item Specification of parameters as RECIPE variables. The term (blame
  PWL not me) is like RETAIN, but made on an instance variable, not on a
  type definition. It allows nomination of non-RETAIN'd parameters to be
  saved that may be written by operator interface or program.
\end{itemize}
```

\subsubsection{On-Line Monitor Mode}

On-Line Monitor mode is equivalent to Instance Configuration mode running on-line.

```
\begin{itemize}
  \item Interface parameters only, defaults to short list
  \item Live values displayed
  \item Interaction dialog for overwriting values
  \item Change from short list to long list
  \item Invocation of block type editor.
  \item Change to type debug mode.
  \item Must be able to capture an On-Line change and make permanent (Eg a
  cold start value).
\end{itemize}
```

\subsubsection{Type Debug Mode}

Type Debug mode is equivalent to Type Edit mode running on-line. The facilities provided are as for a source level debugger. Making changes to the type definition is not possible, but it is possible to inspect (and change) live values from the original block type definition.

```
\begin{itemize}
  \item Interface and internal parameters available
  \item Live values displayed
  \item Interaction dialog for overwriting values
\end{itemize}
```



- \item Invocation of block type editor
- \item Must be able to capture an On-Line change and make permanent (Eg a cold start value).

#### \subsubsection{Template Views and RunTime Panels}

Configuration of the default view on a run-time panel will be via the Template Editor.

#### \subsubsection{Reconfiguring the Template Editor}

The template view should be optionally reconfigurable to the following extent

- \begin{itemize}

- \item Associated parameters can be grouped, with a textual name string for each group.

- \item The order of groups and the order of parameters within groups can be specified.

- \item The display format (Eg Display precision) of parameters can be set. This is in effect setting an attribute that is used by GCT itself (as well as by a runtime system).

- \item Parameters can be nominated as short list parameters. When Template view is initialised in Instance Config or On-Line Monitor mode, only short list parameters are seen. A full parameter list is available from a "View" menu. By default all parameters should be short list parameters (otherwise the user may be confused when nothing happens). [There is a view that the template editor should only ever show the full list, whatever its mode]

- \end{itemize}

#### \section{ST Editor}

##### \label{STEditor}

The ST editor will be a simple text editor, initially based on the XVT facilities, that can be used for either declarations or body editing. ST text will be saved and blocks will be validated by the CDL Object Store. It will be possible for users to install their own editor to use instead of the ST editor.

This editor will allow the textual definition of almost all aspects of a resource from the top layer of a resource down to a parameter attribute. Depending on the nature of the block being edited, the editor may in future assist the user by performing ST syntax checking and by providing templates for the textual definition. (In fact, the ST editor will support the CDL superset of the IEC 1131-3 ST language (see \cite{CDLSpec} and \cite{IECSpec})).

It will be possible to import free format ST text from a file, into a block definition.

In Type Debug mode the following are possible,

- \begin{itemize}

- \item On selecting a piece of text, all variables in the selected

text will be displayed as live values in a pin up window [{\em or pane?}].

- \item (If supported in the target) Breakpoints can be set (and cleared) on lines of code. Single step execution from a breakpoint.
- \item (If selected by the target) Trace of nominated variables at consecutive invocations of the block. Nomination of trace trigger points.

\end{itemize}

A later GCT release should contain an upgraded editor (the 'structured' ST editor) that has an intimate knowledge of ST. E.g., help on syntax, syntax directed colouring, automated indentation, etc..

\section{SFC Editor}  
\label{SFCEditor}  
%\epsffig{sfc.eps}{Example Sequential Function Chart Window}{sfc}

This editor, based on the IEC DIS 1131-3 Sequential Function Chart (SFC) graphical language (see \cite{IECSpec}), will handle the configuration of steps, transitions, and alternate and parallel sequences. It can only be used for POU body editing.

Features of the GCT SFC editor will include:

\begin{itemize}

- \item The SFC editor will present a palette of tool options.
- \item Placing of steps and transitions will be constrained to a grid.
- \item Connecting lines will be constrained to orthogonal polylines. Bends in connections will be constrained to the SFC grid.
- \item A move facility will allow selected steps, transitions or connectors to be moved. Any connections will be stretched.
- \item An object in the work area, eg a step or transition, may be opened for display and editing.
- \item Cut and Paste will be accessible via the menu bar.

\end{itemize}

Tools available in the tool palette will include:

\begin{itemize}

- \item {\bf Pointer :}

The pointer tool will allow selecting or opening one or more steps, transitions or connections.

- \item {\bf Step :}

The Step tool will allow the placing of steps in the SFC work area. A single body may have multiple SFC networks, but each network may only have a single start step.

- \item {\bf Transition :}

The Transition tool will allow the placing of transitions in the SFC work area.

- \item {\bf Connect :}

The connect tool will allow the wiring of steps and transitions.

\begin{itemize}

- \item A step may not be directly connected to another step.
- \item A transition may not be directly connected to another transition.

\end{itemize}

\item Option will be given for routing at connect time. Direct connection will result in auto-routing.

\item Define route from source to destination of connector.

\item Source and destination connection points will be highlighted when appropriate.