

CONTROLLED DISTRIBUTION ONLY IF COLOUR STAMPED

ControlWORKS

CDL Object Manager

APPROVAL FOR DOCUMENT REVISION	Author	Checked	ELECTRONICALLY STORED DOCUMENT	
JOB TITLE	Signature	Date	DIRECTORY PATH	
Senior Software Engineer			pine:/users/cell/jj00/ccase/cdlobj.tex	
			ORIGINATING DEPT: ENGINEERING	
			CONTROLLING DEPT: ENGINEERING	
			CONTROL SHEET	NO. of SHEETS
				24
AUTHOR: John Juer, Jenny Oliver			DOC. TYPE: Software Design Specification	
CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION	ControlWORKS		
	1	CDL Object Manager		
EUROTHERM		EI	DOCUMENT NUMBER	SHT.
© Copyright 1992 Eurotherm Limited			HP024674C301	1

Doc. Revision	Date	Changes
1	October 15, 1992	Initial Draft

1 Scope

This document discusses the design of the Eurotherm ControlCASE (ECC) Object Manager.

The Object Manager handles the creation and modification of ControlWorks objects maintaining coherence between them and acts as a server for ControlWorks utilities. It will allow for transfer of information between editors in an implicit way. For example it will be possible for editors to know if another editor has changed something they are viewing.

The Object Manager will initially be a set of C or C++ routines that can be called by ControlWorks utilities to view, edit and manage CDL objects.

Initially the Object Manager will be linked in to the editors as one application; ultimately the Object Manager will be a separate process communicating by RPC and/or X client protocols (on Unix) and DDE and/or DDL (on Windows) to editor processes.

Initially (in its linked form) the object manager will be re-entrant; in its server form the Object Manager will be multi-threaded to provide load sharing between its clients.

2 Overview

CDL objects are units that organise parts or all of control algorithms expressed in CDL (or a graphical equivalent such as sequential function charts, function block diagrams or spreadsheets). CDL objects have declarations of data and other object instances that they will manipulate, and one (or more) algorithms that do the manipulation. (This is defined precisely in section 2.1.) In general the CDL object manager will provide an interface that works on these "units of organisation" directly, even though they can be created implicitly by CDL. The algorithms that manipulate the object will however in general be dealt with by providing to the object manager text buffers containing fragments of CDL (e.g assignment statements, expressions etc.).

The CDL Object Manager will have to

- Maintain a RAM resident definition of CDL objects for fast access
- Store and retrieve definitions of CDL Objects, including any associated graphic information (e.g from SFC diagrams) from disk
- Provide facilities for editors to interactively create CDL Objects, store them to disk, delete them from disk and/or from the RAM database
- Provide facilities for back end processors to read CDL Objects and translate them to product specific form. (For example T1000 GRF files)

A primary goal for the design of the Object Manager is that disk store for CDL objects is itself CDL. There are various reasons for this —

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON FROL SHEET.	DOCUMENT REVISION 1	ControlWORKS	
		CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER	SHT.
		HP024674C301	2

- To maintain portability (especially for customers between Windows, Unix etc.), flexibility and make upwards compatibility more easily achievable.
- To make it easier to write back end processors that take CDL, and create both an executable control strategy and graphics to represent that strategy on a controller (e.g T1000 and the GRF file). Such back end processors could make use of the object manager services to read the CDL file.
- To make it feasible to construct default pictures from raw ST
- To centralise management of files representing CDL objects and minimise the number of file types and hence
- To promote reuse of code and facilities

Ideally editors (and back end processors) should not have to "understand" or generate CDL. In this way if CDL changes editors will continue to work, providing the interface to the object store does not change.

In addition the Object Manager will provide various ways of manipulating CDL objects. Objects and their declarations (instantiations of sub-objects) may be defined using a procedural method, as in C routines such as (this is just an example, and not an actual proposed set of C routines):

```
obj_handle = create_new_function_block("A_new_block");
ok = add_declaration(obj_handle, "in1", DINT, NULL, NO_BOUNDS, INPUT);
ok = add_declaration(obj_handle, "in2", DINT, NULL, NO_BOUNDS, OUTPUT);
ok = add_declaration(obj_handle, "ablock", FBLOCK, "block_type1", NO_BOUNDS, INTERNAL);
```

Other interface calls will exist that take in CDL text to define the algorithms that manipulate or initialise declarations. For example

```
ok = add_cold_start_expr(obj_handle, "in1", " 1 * 10 - otherin");
```

where the first parameter is the object the cold start value is being added to, the second the parameter within the object being given the cold start and the third a buffer containing a CDL expression - the cold start expression.

It could also be possible to define a set of declarations, e.g input/output parameters, by sending the declaration section for the CDL.

Some editors will need to analyse CDL fragments quite extensively; for example an FBD editor may need to embed in the CDL algorithm describing the FBD diagram, information that will allow it to restore the user's FBD diagram; the object manager will provide services to query parse trees of CDL, and for editors to embed and retrieve information in CDL fragments.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 3

2.1 CDL

The CDL language consists of a set of Program Organisation Units which naturally form part of the set of objects that need to be handled by the Object Manager.

POUs are Function Blocks, Programs, and Functions, (later on they may include stand alone Services and Handlers).

Program Organisation Units are arranged either in a Library or a Project. Libraries may be shared between other libraries and projects. A Project contains a private library for an application.

In addition to these POU's there are two other sorts of objects in CDL, namely Resources and Configurations. Resources and Configurations exist within a Project but not within libraries they are not reusable; so they do not form part of the project library itself.

Resource definitions are, however, are also POU's; they can be defined in FBD, or ST (by wiring together block declarations).

POUs have a declaration section, divided into an interface specification (the inputs, outputs and in outs), and the internals. (Resource definitions just have internals).

POUs are defined either in ST, FBD or SFC (or possibly in another format e.g spreadsheet translated by editors to CDL, or even in another language such as C).

SFCs and FBDs contain CDL objects which have associated ST, FBDs or SFCs — in other words these objects have associated with them the body of a POU but not the interface (input, output and in-out) declarations associated with a POU. (This could be thought of as having an empty interface) They may have internal declarations, however. Objects in this class and the associated "POU body" are —

Transitions have an associated ST ¹ expression

Actions have an associated FBD, SFC or ST body ²

Declarations have an associated cold start ST expression

Furthermore certain CDL objects will require the maintenance of attributes which will differ from editor to editor. For the SFC editor, for example, Steps, Transitions and Actions will require graphic information relating to their position in the SFC picture to be maintained.

Attributes will be embedded in CDL as strings surrounded by { and }. The strings will specify a set of name, value pairs. for example ³

```
PID1 : PID { X_pos := 300, Y_pos := 500 }
```

The set of objects (to which attributes may be attached) are, roughly speaking, all instanced objects including steps, actions and transitions, and an object that represents the definition of the POU. It may be necessary to attach attributes also to parameters of instanced function blocks.

¹ Later this may be defined in FBD or Ladder as per IEC

² In IEC Actions can have a boolean output, currently we do not support this

³ The syntax for attributes is yet to be decided; this follows syntax currently in use by the ST compiler

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 4

2.2 Object manager facilities

The Object Manager will support Project and Library management as follows —

- Provide services to list, open, copy, rename, build and delete projects and libraries. Note that in the first instance these commands will be mapped onto simple shell/batch commands such as those in the FBB.
- Provide services to delete, add and copy blocks to/from libraries and projects.
- Provide services to explore blocks in a library/project by fast load (i.e not loading the body or attributes necessary to show graphics), and examination of their components.

The Object Manager will support editing of blocks or components of blocks that have an associated body as follows —

1. The CDL definition of a block (and all its components) can be “loaded”. This makes the block ready for editing.
2. For all components that have a body the Object Manager will return the name of the editor the component was created with.
3. If the component's body was created textually an ST buffer is available for editing the body. The Object Manager will provide services to parse and verify this buffer.
4. If the component's body was created graphically then the CDL object's and their logical CDL interconnection may be explored, (see below).
5. Any CDL object has attributes, which can be modified and used to store graphical lay out (or any other editor specific) information. The CDL Object Manager will provide facilities to add a name,value attribute pair, retrieve a named attribute's value, and delete attributes by name. This will probably be provided as a library of routines available to each editor, rather than via the object manager interface.
6. The Object Manager will provide services to transform a ST buffer into a set of “parse tree objects”⁴. Services⁵ will be defined allow
 - querying of the tree, for example to find if it corresponds to a valid tree for the editor — a FBD editor could not handle a tree that contained IF statements say.
 - adding/deleting sub-trees to the parse tree objects; for example an FBD editor could interactively create the tree corresponding to a wiring diagram by “editing” a parse tree of wiring expressions.
 - adding of information to ST parse trees that is stored and retrieved by the manager. For example an FBD editor could decorate a parse tree corresponding to the wiring with information about routing etc.

⁴Since the first release of the tools will support SFC and ST only it is unlikely these will be required immediately

⁵An alternative is a set of library routines that editors could link in

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON TROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
	EI	DOCUMENT NUMBER HP024674C301	SHT. 5
EUROTHERM © Copyright 1992 Eurotherm Limited			

2.3 Adding attributes and editing SFC parse trees

When describing CDL it is possible to talk about SFC objects being "connected", for example a step is connected to a transition. In order to understand and manipulate CDL there will be internally in the object manager a representation of this. It could be possible to ask the object manager which steps are connected to which transitions and vice-versa. If this is of general use to any editor then the object manager should support such services.

This is not any generalised notion of graphical connectivity, but rather an explicit representation of CDL connectivity. Graphical connectivity will require much more information than the mere fact that a step is connected to a transition; for example the path of the connected wire.

The Object Manager stores this information as "attributes" of the CDL object. The Object Manager can have no understanding of these attributes since in general the attributes will depend on the editor being used — for example a spreadsheet editor will have a completely different set of attributes to a SFC editor, but both will be defining CDL steps and transitions.

It is important to make sure that the way attributes are associated with objects, and the objects they can be associated with, is of general use to all editors.

Certain editors may need to add "attributes" at a finer level than the objects that make up a set of declarations. For example an FBD editor may want to create the parse tree corresponding to a wiring expression, and add attributes to the parse tree that correspond to the graphical lay out of the wiring expression. The Object Manager will also have to support storage and retrieval of these attributes.

Below various alternative strategies for attribute storage and retrieval are considered with respect to possible editors.

2.3.1 SFC editor

The CDL for an SFC does define implicitly connectivity in terms of steps and transitions. However graphically a transition is not a simple line from one step to another, because of divergence of a step into a set of parallel activations, convergence of the same, selection of one out of a set of alternate steps, and convergence of a set of selected steps into one step. Graphically could be thought of as intermediate objects which connect to steps. Textually an selection is represented by several transitions specified in more than one place.

It may therefore be more natural to graphically represent divergent connections between steps as connections from a step to an intermediate object which may be a parallel branch or a selection, and then connections to the destination steps, and convergent connections as the opposite to this.

The CDL object whose attributes represent this intermediate object is the step that is the point of divergence or convergence⁶. The attribute for a transition can then be simply describe the line from the intermediate object to the step.

Having analysed the CDL the object manager knows whether a step has a branch or selection transition out of it, or into it, and therefore can tell the SFC editor whether or not to expect the intermediate object.

Consider figure 1, which defines some steps and transitions for the following SFC picture and shows x y coordinates of some key points with 0,0 in the top left of the picture.

The CDL used to store this including attributes could be —

⁶ In the case of a transition that is both rendezvous and a parallel activation then there are two intermediate objects merged into one. An alternative and probably better strategy would be to make the transition the intermediate object. The aim of the present discussion is simply to validate the approach, not define the way that the SFC editor should use attributes. That is for another document!

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
	EI	DOCUMENT NUMBER HP024674C301	SHT. 6
EUROTHERM © Copyright 1992 Eurotherm Limited			

```

(* 'xy' gives the top left hand corner of the step *)
STEP x1 { sfced := 'xy:9,2' } : x1_act(N); END_STEP

(* simple transition from one step to another *)
TRANSITION { sfced := 'start:12,4 end:12,7' }
    FROM x1 TO x2 := 1; END_TRANSITION

(* transition out of step has a parallel activation.
'branch' defines where to position intermediate object *)
STEP x2 { sfced := 'xy:9,7 branch:4,11,22,11,12,9' : x2_act(N); END_STEP

(* one CDL transition object, but two connections from intermediate
object to steps *)
TRANSITION { sfced := 'start1:8,12 end1:8,13 start2:18,12 end2:18,13' } FROM
    x2 TO ( x3, x4 ) := 1; END_TRANSITION

STEP x3 { sfced := 'xy:5,13' } : x3_act(N); END_STEP

STEP x4 { sfced := 'xy:15,13' } : x4_act(N); END_STEP

(* one CDL transition object, but two connections from intermediate
object to steps *)
TRANSITION { sfced := 'start1:8,15 end1:8,17 start2:18,15 end2:18,17' }
    FROM ( x3, x4 ) TO x5 : x5_act(N); END_STEP

(* step is reached via a rendezvous, 'rendezvous' defines where to position
intermediate object *)
STEP x5 {sfced := 'xy:9,20 rendzevous:10,20,12,20' } : x5_act(N); END_STEP

(* This transition object stores where the line from
the intermediate selection object to the destination
step is drawn *)
TRANSITION { sfced := 'start:10,24 end:10,26' }
    FROM x5 TO x6 := 1; END_TRANSITION

(* This transition object stores where the line from
the intermediate selection object to the destination
step is drawn *)
TRANSITION { sfced := 'start:15:24 pt1:15:26 pt2:25,26 pt3:25,1 pt4:12,1 end:12,2' }
    FROM x5 TO x1 := 1; END_TRANSITION

(* This step stores where the intermediate selection object is drawn *)
STEP x6 { sfced := 'xy:7,26 select:12,22,12,24' } : x6_act(N); END_STEP

```

Here the editor defines one attribute for each step and transition which can be retrieved to draw the graphic for each connection between steps and transitions. All other information is implicit in the CDL — in other words the editor asks the object manager which steps are connected to which transitions. Before the editor can draw a transition (and indeed understand the attribute string for the transition) it must find out whether the transition is a branch or rendezvous or selection or simple transition. Again all this information is already implicit in the CDL.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON TROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 7

While the picture is being edited the editor performs transactions with the object store such as "add step", "define transition".

One possible problem is how to store incompletely defined transitions, i.e. that do not go anywhere or do not come from anywhere. This implies the object manager must understand partially incorrect CDL, such as

```
TRANSITION { sfcd := 'start:10,22 end:10,26'} FROM x5; END_TRANSITION
```

The object store would also provide a facility to verify that an SFC was correct.

The object store would provide the means to dump the SFC to file.

An alternative to this is that the editor stores a full representation of the SFC as a large attribute of the initial step, or as less large attributes of the set of steps. In this case the CDL object manager does not need to provide facilities to query CDL step connections and the link between the editor and the object store is not so intimate, but there is the question of how the CDL is generated — it means editors have to understand how to generate valid CDL. If CDL changes so does every editor.

2.3.2 FBD editor

The CDL representation of FBD diagrams will use CDL wiring expressions⁷, since general IEC "wiring", i.e. statements containing function block input assignment, may contain other things not expressible in FBD e.g. loops, case and if statements, and may vary the expression assigned to an input on different call of the block. Any CDL block which just contains CDL wiring will be expressible as an FBD.

For the FBD editor CDL wiring expressions will require attributes to be maintained. For example consider the wiring expression

```
VAR
  I: PID ( PV ::= 10.0 + X1.PV );
```

In an FBD diagram this consists of wiring of X.PV to the output of an add function, which has two inputs, 10.0 and X1.PV.

This representation of wiring means that wiring expressions are associated with the declaration of the block "pulling" the expression, i.e. the ultimate destination of the wire. This is correct since FBDs support fan out but not fan in.

There are three possible ways the graphic attributes could be stored. These are

1. There is one attribute string attached to the declaration of the block which gives the information needed to draw the wire, i.e.

```
X: PID ( PV ::= 10.0 + X1.PV ){ all attribs for block and wires};
```

2. There is one attribute string attached to the input of the block —

⁷This refers to the extension to IEC where a statement in an instantiation of a block such as X: PID (PV ::= 10.0 + X1.PV); is a "wire" i.e. PV cannot be assigned to elsewhere and the expression is evaluated on every call of X

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON FROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
	EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301 SHT. 8


```
X: PID ( PV { attrib string for wire } ::= 10.0 + X1.PV )
      {attribs for block X};
```

Note we now need one attribute string for each wired input, and one for the block declaration.

1. There is one "attribute" string attached to every sub-expression of a wiring expression, giving simply the position of single wires between inputs and outputs of functions and function blocks and the position of functions —

```
X: PID ( PV ::= 10.0 { attrib for 10.0 }
      + { attribs for + }
      X1.PV { attribs} )
      {attribs for block X};
```

This implies that there is a text buffer that contains the CDL wiring expression and CDL attributes inserted by the FBD editor. The object manager provides services to temporarily turn this buffer into a parse tree and allows queries on the tree for attributes.

The trade offs are

1. The amount of work the editor has to do in understanding the attribute strings, to recover connectivity information already implicit in the raw CDL
2. The complexity and amount of interface between the editor and the object manger.

Below are some guesses as to what the attribute for the fbd editor might look like.

First with option 1

```
VAR
I { edpos:= '30,40' } : PID ( PV ::= 10.0 + X1.PV )
  { fbd := 'FnObj(Id2,+,10,20),Conn(X.PV,Id2.1,20,25),Conn(10.0,Id2.2,15,30)' }
```

there is almost another language for the FBD in the attribute string.

Secondly with option 3 we might have,

```
VAR
I { edpos:= '30,40' } : PID ( PV { fbd := 'wire:20,10:30,10' } ::= 10.0
      {fbd := 'wire:15,30:20,30 ' } +
      { fbd := 'pos:10,20:15:40 wire1:10,20:30:50 wire2:32,40:45,50' }
      X1.PV { fbd := 'wire:20,25' } )
```

Here the editor has to work with a parse tree of the wiring expression in order to discover connections between functions and function blocks. The attributes of each object in the object store include the positions of input wires.

The editor and the object store have a more inter-twined interface.

Transactions would allow

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 9

- creation or deletion of function call nodes (or direct wiring)
- connection of function block outputs to inputs of function calls
- connection of function call outputs to inputs of function calls
- connection of function call outputs to function block inputs

In order to allow incomplete wiring the editor might have to create "non-existent" dummy objects to wire to)

These transaction effectively edit a CDL parse tree of expression.

For example

```

/*
  pin1-|-----|
        |  ADD  |
  pin2-|         |---pin3
  obj2-|         |
        |-----|
*/

obj1 = create_fcall("ADD",attributestring);
ok = add_conn_to_input(obj1,PIN,"IN1",,pin1);
ok = add_conn_to_input(obj1,PIN, "IN2",pin2);
ok = add_conn_to_input(obj1,OBJECT,"IN3", obj2);
ok = connect_obj_to_pin(obj3,pin3spec);

```

2.3.3 Spreadsheet editor

Spreadsheets are a representation of a subset of SFCs. Each column in the spreadsheet defines a step. Each row is a variable. Each cell contains an expression which is assigned to the variable when the step is active.

At the bottom of the spreadsheet are groups of pairs of special rows that define transitions, labelled "ON" and "GOTO". The first row in the pair is the "ON" row, the second the "GOTO" row.

Cells in the "ON" row contain expressions that cause a transition from the step in the corresponding column to the step named in the second "GOTO" row.

The "ON" and "GOTO" rows define either simple transition or selections from one step to the next step.

The other cells containing expressions assigned to variables define ACTIONS active when a step is active. In order to reproduce the spreadsheet the editor needs to recover the variables assigned to in ALL the actions (some of the cells may be empty indicating that no assignment is made to the variable). It also needs to recover the individual expressions for each action.

Now the variables are declared in the declaration section for the block. Therefore attributes can be attached to the declaration section which indicate the row position in the spreadsheet of those variables. Similarly attributes declared for each step can define the column position of the step.

There could be various ways of dealing with the expressions in the cell.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 10

1. The action has a CDL text buffer containing the assignment statements, and an attribute giving the cell number to line number (in the text buffer) correspondence. This allows retrieval of the assignment statement from the text buffer by the editor. The editor has to be able to understand the syntax of the assignment statement.
2. The action has a CDL text buffer. The object manager provides services to turn this buffer into a parse tree objects and allows queries on the tree for attributes, and edits on the tree, for example

```
ok = replace_assign_statement(stmt_handle,"IN","A - 10 * 3");
```

might be the way the editor replaces a cell assigning expression A - 10 * 3 to IN.

As for the FBD editor this illustrates that either the editor stores more complex attributes in CDL objects, or it embeds attributes into CDL text and has a more inter-twined interface with the CDL object manager.

2.4 Invalid CDL

Editors must be able to store partially complete definitions.

If CDL is the store for each editor then there are some not mutually incompatible alternatives

1. The CDL object store must understand an "invalid" set of CDL that has been defined by an editor
2. The editor must interpret a valid piece of CDL into a incomplete diagram.

When dealing with SFC the first option is usable, since the "invalid" CDL will consist of non-connected or partially connected CDL components. These are relatively easy to express in CDL, viz:

```
TRANSITION FROM x5; END_TRANSITION
```

Wiring expressions (such as in FBD) are more problematic for the object manager. It is probably preferable that the editor creates special variables to wire to that it knows are not really present.

Another issue when providing services to parse CDL is defining what CDL fragments are related in what context. A function block body would not be valid when trying to declare a set of input variables.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited		EI DOCUMENT NUMBER HP024674C301	SHT. 11

3 Phase 1 Object Manager Interface

Phase 1 is not going to support FBD or spreadsheet editors. Therefore only a subset of the functionality above is required.

In Phase 1 there will be ONE executable with all the editors and the object manager linked together.

The interface to the object manager will be using a set of C++ objects. In order to allow the implementation of the objects to change, without requiring any recompilation in code using the interface every interface object will be implemented with no private data or methods, except one anonymous pointer to the actual implementation, viz

```
// Declare the anonymous class
class RealInterfaceExample;
class InterfaceExample
{
private:
    RealInterfaceExample* it;
public:
    void Amethod();
};
```

Note this will have a slight disadvantage in that there can be no in-line methods in the class (since the "real" object cannot be used in the header). It should, however, make it easier to split the interface between processes later on.

The first step in using the object manager is to create an interface object of class "ObjStoreInterface". This will ultimately be used to manage such things as network connections to the object manager.

The interface object will also be passed an *ErrorStream* pointer *ErrorStream to be defined* which will be used for passing global error messages from the object manager back to the editors (e.g no more disk space, cannot write file etc.).

The interface object has a method used to register an editor object for each editor. When the object manager opens a block for editing it will find an attribute which is the editor used to create the block; it will then use the interface object to look up the editor that edits the block and call back that editor.

The interface object also contains methods to list projects, libraries, and to open or create projects or libraries.

Projects are viewed as specialisation of Libraries. For example a Resource can only be created in a Project. So there is a library parent class with a child Project class.

Basic information about language objects is exchanged using the "CDLInfo" structure.

```
// CDLInfo used to declare a new object
struct CDLInfo
{
    CDLType type;
    CDLMode mode;
    char*   name;
    char * typename;
    bool    isRef;
    ArrayInfo array;
};
```

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
	EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301

```

class ObjStoreInterface
{
public:
    char*   nameOfObjectMgr; // will be cms address of Object Manager
    ErrorStream* GlobalErrors;

    ObjStoreInterface (char* name,
                       ErrorStream* errors);

    // Listing Projects, Libraries
    StringListIterator ListAllProjects();
    StringListIterator ListAllLibraries();

    // Open Project or Library
    Library* OpenLibraryOrProject (char* name);

    // Create Project or Library
    Library* CreateLibraryOrProject (char* name,
                                     StringList* LibraryList, // library path
                                     bool IsProject);

    // Register an editor that can be invoked by the Object store
    bool RegisterEditor (Editor* theEditor);

    // Find out which editors the Object store knows about
    EditorListIterator* ListEditorList();
};

```

```

class Editor
{
public:
    // Will link editor via an attribute to the block
    const char* identifier;

    // Method to be called to edit something.
    // Should return immediately having fired up the application
    // and initialised the event loop etc.
    bool EditBlock (IndependentEditable* theBlock, bool brandNewBlock);
};

```

```

struct BlockInfo
{
    char* name;
    char* libraryName;
    BlockType type; // program, function_block , resource
    BlockTarget target; // FMC, PO, PC3000 - to control builds
};

```

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 13

```

class Library
{
public:
    // For convenience returns the interface object
    CMLObjStoreInterface* GetContext();

    // is it a project or a library?
    bool IsProject() {return FALSE;}

    // Copy Project or Library (does an implicit create, error if
    // already exists unless 'merge' is true in which case an
    // attempt is made to merge the projects/libraries )
    bool CopyProjectOrLibrary (char* to, bool merge);

    // Rename project - errors go to the global error stream
    // true returned on success, false otherwise
    bool Rename (char* to);

    // Delete the Project or Library.
    // errors go to the global error stream
    // true returned on success, false otherwise
    bool Delete ();

```

A "CDLLangElement" is the base class that represents one of a function block type, function, function block instance, program type or instance, a variable (including an array variable), a step, transition, service, action, task or resource.

Each CDLLangElement has an associated CDL buffer, whose use depends on what type the object represents and how the object is defined. For example a simple variable may have a cold start expression in its buffer, a transition the transition condition, a function block type declared in text the whole of the function block body, an action the action body. However a step will not have a CDL buffer.

Every CDLLangElement also has a set of methods to add attributes or property definitions (the latter can only be applied to var references).

A specialisation of CDLLangElement is the IndependentEditable class which represents CDL objects that can be edited by an independent editor. These are function block types, functions, programs, resources, actions, and services.

Each IndependentEditable has a set of CDLLangElements associated with it, which are the CDL objects declared in its scope. Objects are declared using AddElement and deleted using DeleteElement.

Sub-objects are retrieved from any CDLLangElement using the GetElement method. This is because, for example, it is possible to instance a function block which has an already defined set of sub-objects, its parameters. Similarly a step has the step flags.

Any IndependentEditable can be placed on-line, by giving the address (the full Resource path name) to the actual object. At this point any sub-object can have its on line values queried using the GetLiveValue method.

Need to consider download, start stop i.e REX interface

A Block object is a further specialisation of an Independent Editable. It represents something that has its own definition i.e a Program, Resource or Function Block definition. It can be saved (with or without verify), built, unbuilt or renamed.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 14

// Library object continued

```

// Build Library, errors go to the passed in error stream
// true returned on success, false otherwise
bool Build (ErrorStream*);

// errors go to the global error stream
// true returned on success, false otherwise
bool UnBuild ();

// LibraryList is the list of libraries used by this library.
// LibraryList management --- Add,Delete & change all do UnBuilds
// errors go to the global error stream
// true returned on success, false otherwise
bool AddToLibraryList (char* name, int position);
bool DeleteFromLibraryList (int position);
bool ChangeLibraryListOrder (int from, int to);

// Returns the list of used libraries
StringListIterator* ListLibraryList();

// Block Management
BlockInfoListIterator* ListBlocks(BlockInfoSet& options);

// Only blocks which match options are included, so that can search
// for a named set of blocks
BlockInfoListIterator* ListBlockDependencies (char* blockName,
                                              BlockInfoSet& options);

// find which Library a given block is in, including
// this library
char* LocateBlockInLibraryList (char* name);

// Block creation etc. Will fail for Resources
// since this is a Library object
bool CreateBlock( BlockInfo& Options);
bool DeleteBlock (char* name);

// Kick off an appropriate edit session.
// returns true if it succeeds, errors go to
// global error stream
bool EditBlock( char* name);

// Copy block into this library from another
// returns true if it succeeds, errors go to
// global error stream
bool CopyBlockFromLibrary (Library* fromLibrary,
                          char* fromBlock);
};

```

CONTROLLED DISTRIBUTION COPY T.Y IF COLOUR STAMPED ON FROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 15

```

class Project : public Library
{
    // A project contains a Library, and can also have
    // Resource (and later Configuration) 'blocks'

public:
    bool IsProject() {return TRUE;}
};

```

A Step requires special treatment and is a specialisation of a CDLLangElement. A step has methods to add and delete from its action list (limited to one action for now), and to retrieve the actions. Note that addition of actions is by name, so the action can be separately defined. It also has methods to get the transitions to and from the step. A step has no CDL buffer.

A Transition is another specialisation of a CDLLangElement allowing to and from steps to be added and retrieved. Note that addition of these is by name so the steps can be separately declared.

Steps and transitions can be retrieved from the parent CDLLangElement by name.

3.1 Memory Management

Deleting objects has no side effect such as saving, or validating any associated CDL; there are explicit methods for any of these operations; deletion is purely for memory management.

Deleting the ObjStoreInterface object will cause all memory allocated by the object manager to be freed. All pointers to any interface objects will be invalid.

Deleting a Project or Library object will cause all memory associated with that object to be freed, so all pointers retrieved from that object will be invalid.

Deleting a Block object will cause all memory allocated with that object (i.e all sub-objects) to be free. Any pointers retrieved from the Block object will be invalid.

No other objects should be deleted by editors.

4 Examples of using the interface

This section tries to outline how to use the objects defined above to perform various standard functions.

Every process using the object manager must have one (or more?) ObjStoreInterface object.

4.1 Library and Project management

4.1.1 Listing Projects, Libraries

Use the ListAllProjects or ListAllLibraries method on the ObjStoreInterface object.

CONTROLLED DISTRIBUTION COPY TRY IF COLOUR STAMPED ON FROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 16


```

class CDLLangElement
{
public:
    const CDLInfo info;                // e.g. Function Block

    // CDL fragment can ColdStart, Transition, or POU body
    virtual CDLBuffer* GetCDLBuffer ();

    // returns true if successful. Errors go
    // to the error stream of the associated IndependentEditable
    // object.
    virtual bool ReplaceCDLBuffer (CDLBuffer *);

    // Applies to any CDL object e.g verifies a function
    // block body, a cold start value, a SFC. Errors
    // reported in error stream of owning independent
    // editable.
    virtual bool Validate ();

    // attribute / property handling
    char* GetAttrOrPropValue (char* name, bool& isAttr);
    bool ReplaceAttrOrPropValue (char* name, char* value);
    bool DeleteAttrOrProp(char* name);

    // On-line
    const SetOfValue* GetLiveValue();

    // return the Element for a dependent element
    // e.g a parameter of the block, or block
    // instance
    CDLLangElement* GetElement (char* name);

    // Find the set of elements
    CDLInfoListIterator* ListElement (CDLInfoOptionsSet options);

};

```

CONTROLLED DISTRIBUTION COPY
ONLY IF COLOUR STAMPED ON
CONTROL SHEET.

DOCUMENT
REVISION
1

ControlWORKS

CDL Object Manager

EUROTHERM

EI

DOCUMENT NUMBER
HP024674C301

SHT.

17

© Copyright 1992 Eurotherm Limited

```

// smallest stand alone edit-able object
// i.e. block function action service resource

class IndependentEditable : public CDLLangElement
{
public:
    const CDLBodyType bodyType; // e.g. SFC body
    ErrorStream* errorStream;

    bool AddElement (CDLInfo& info);
    bool DeleteElement (char* name);

    // kick off separate edit session (e.g. for an ACTION inside a SERVICE)
    bool EditElement (char* name);

    Step* GetStepElement (char* name);
    Transition* GetTransitionElement (char* name);

    // On line
    bool OnLine(char* address);
    bool OffLine();
}

```

```

class Block : public IndependentEditable
{
private:

public:

    bool Rename (char* name);
    bool Build (ErrorStream* );
    bool UnBuild (ErrorStream* );

    bool Save (bool Verify);
};

```

CONTROLLED DISTRIBUTION COPY
ONLY IF COLOUR STAMPED ON
TROL SHEET.

DOCUMENT
REVISION
1

ControlWORKS

CDL Object Manager

EUROTHERM

© Copyright 1992 Eurotherm Limited

EI

DOCUMENT NUMBER
HP024674C301

SHT.

18

```

class Step : public CDLLangElement
{
public:
    bool AddToActionList (char* name,
                        ActionQualifier qual, int position);
    bool DeleteFromActionList (int position);
    /* LATER
    bool ChangeActionListOrder (int from, int to); */
    bool SetActionQualifier (char* name, ActionQualifier qual);
    ActionListIterator* GetActionList ();

    StringListIterator* GetTransitions (bool from /* else to */ );

    // There is no buffer for steps
    virtual CDLBuffer* GetCDLBuffer () {return NULL;}
    virtual bool ReplaceCDLBuffer (CDLBuffer *) {return FALSE;}
};

class Transition : public CDLLangElement
{
public:
    bool SetToSteps (char** name);
    bool SetFromSteps (char** name);

    StringListIterator* GetSteps (bool from /* else to */ );
};

```

4.1.2 Open Projects or Library

Use the OpenLibraryOrProject of the ObjStoreInterface object passing the name of the library or project, and retrieving a pointer to a library or project object.

4.1.3 Creating a Project or Library

Use the CreateLibraryOrProject method of the ObjStoreInterface object passing the name of the new library or project, a bool to indicate it is a project and the list of shared libraries. The method returns a new library or project object.

4.1.4 Copy Project or Library

Use the CopyProjectOrLibrary method on a Library object passing the name of the new library and a bool to indicate whether or not merging with an existing library is allowed.

4.1.5 Rename Project or Library

Use the Rename method on a library object, passing the new name.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 19

4.1.6 Delete Project or Library

Use the Delete method on the library or project object.

4.1.7 Build Project or Library

Use the Build method on the Library or Project object, passing an ErrorStream for build error reporting.

4.1.8 UnBuild Project or Library

Use the UnBuild method on the Library or Project Object.

4.2 Block Management

4.2.1 Find Block from Type Name

Use the LocateBlockInLibraryList method on a Library object passing the type name of the block. The return is the name of the library the block is in (which allows the library object to be retrieved and then the associated block object).

4.2.2 List Blocks in current Project

Use the ListBlocks on a Library object, passing in an empty options set.

4.2.3 List Blocks given Project, Library

Retrieve the Library/Project from the ObjStoreInterface object and then list its blocks.

4.2.4 Create Block

Call the CreateBlock method for the Library/Project giving the options for creation (e.g the type of the block, (Resource, Program, Function Block) and the name of the editor for the block.

4.2.5 Copy Block from another Library

Call the CopyBlockFromLibrary on a library or project object.

4.2.6 Rename Block

Only blocks that are being edited can be renamed; the editor is given the Block object after an EditBlock call on the Library/Project object, and then the Rename method on the Block object can be used.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON TROL SHEET.	DOCUMENT REVISION 1	ControlWORKS	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	CDL Object Manager	
		DOCUMENT NUMBER HP024674C301	SHT. 20

4.2.7 Open Block for browsing template

Not supported at present, the block must be opened for editing.

4.2.8 Delete Block

Blocks that are being edited can be deleted; the editor is given the Block object after an EditBlock call on the Library/Project object, and then the Delete method on the Block object can be used. Alternatively the DeleteBlock method on the Library/Project object can be used.

4.2.9 Build Block

Only blocks that are being edited can be directly built; an editor can call the Build method on a Block object. Alternatively the whole of a Project or Library can be build.

4.2.10 Get a Block's dependencies

Given a block name and a set of options to match (e.g look for only Function Blocks) the ListBlockDependencies method on a Library/Project will return the block information about blocks a block depends on.

Should this be only direct dependencies? Or indirect?

4.3 Block Editing

4.3.1 Open Block for editing

Call the EditBlock method on a Library/Project. The appropriate editor will be invoked using the Editor object store in the ObjStoreInterface object.

4.3.2 List VARS (VAR_IN, VAR_OUT, VAR_INOUT, VAR, SERVICES)

Use the ListElement method with associated options on any CDLLangElement object. CDLLangElement objects are retrieved by name from any CDLLangElement object (and hence Block, Step, Transition and IndependentEditable object).

4.3.3 Delete Interface element

Call the DeleteElement method on an IndependentEditable object, passing the element name.

4.3.4 Add Interface element

Call the AddElement method on an IndependentEditable object, passing a CDLInfo structure.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON TROL SHEET.	DOCUMENT REVISION 1	ControlWORKS	
		CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 21

4.3.5 Save block

Call the Save method on a Block object.

4.3.6 Open body for editing

Any IndependentEditable (and hence Block) may have an element that itself can be edited, by calling the EditElement method passing the name of the element to be edited.

4.3.7 ST editing

Any CDLLangElement object apart from steps and any action, service or function block whose body is defined as an SFC, has an CDL text buffer that can be directly retrieved for text editing.

4.3.8 Get ST buffer

Call the GetCDLBuffer method on a CDLLangElement object.

4.3.9 Replace ST buffer

Call the ReplaceCDLBuffer method on a CDLLangElement object. The old buffer will be deleted.

4.3.10 Validate ST buffer

The Validate method on any CDLLangElement will parse and Validate any associated CDL buffer.

4.3.11 Attributes

Any CDLLangElement object has a set of attributes that can be manipulated. Object that are VAR REFERENCES additionally have a set of built in properties.

4.3.12 Add attribute

Call the ReplaceAttrOrPropValue method on a CDLLangElement object.

4.3.13 Delete attribute

Call the DeleteAttrOrPropValue method on a CDLLangElement object.

CONTROLLED DISTRIBUTION COPY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS	
		CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER	SHT.
		HP024674C301	22

4.3.14 Find attribute

Call the GetAttrOrPropValue method on a CDLLangElement object, passing the name of the attribute to get its string value.

4.3.15 SFC Editor

Steps and Transitions are objects that can be retrieved from IndependentEditable objects whose body is defined by an SFC.

4.3.16 List STEPs in SFC

Call the ListElement method in a CDLLangElement object passing options to ensure only steps are selected.

4.3.17 List TRANSITIONs

Call the ListElement method in a CDLLangElement object passing options to ensure only transitions are selected.

4.3.18 Verify SFC body

The Validate method on a IndependentEditable will validate any SFC definition, (along with action bodies etc).

4.3.19 Add Step

Call AddElement on a Independent Editable object, declaring it as a step.

4.3.20 Get Initial Step

Search for an initial step type using the ListElement method on a CDLLangElement.

4.3.21 Delete Step

Call DeleteEkement on an IndependentEditable object.

4.3.22 Change Action modes

Retrive the Step object by name using the GetStepElement method on an IndependentEditable object, and then use the SetActionQualifier method on the Step object.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS	
		CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 23

4.3.23 Get Actions for Step

Call the GetActionList method on a Step object. Note this returns a set of names and qualifiers.

4.3.24 Get Transitions from Step

Call the GetTransitions method on a Step object, passing the bool parameter with value true. Note this returns a set of names.

4.3.25 Get Transitions to Step

Call the GetTransitions method on a Step object, passing the bool parameter with value from. Note this returns a set of names.

4.3.26 Get to/from Steps for Transition

Transitions are retrieved from a IndependentEditable object using the GetTransitionElement method. From a transition object the to/from steps are retrieved using the GetSteps method.

4.3.27 Add transition

Transitions are declared using the AddElement method on a IndependentEditable object.

4.3.28 Change transition

The CDL buffer for a transition can be edited using the GetCDLBuffer method on a CDLLangElement object. Transitions may be deleted using the DeleteElement method on an IndependentEditable object.

4.3.29 Add Action

Actions are declared using the AddElement method on a IndependentEditable object.

4.3.30 Delete Action

Actions are deleted using the DeleteElement method on a IndependentEditable object.

4.3.31 Cold start values

Any CDLLangElement that may have its cold start value set has a CDL buffer which can be edited using the GetCDLBuffer method on a CDLLangElement object.



CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	ControlWORKS	
		CDL Object Manager	
EUROTHERM © Copyright 1992 Eurotherm Limited	EI	DOCUMENT NUMBER HP024674C301	SHT. 24

