Clovis Solutions, Inc.

ClovisWorks User Guide

Release 2.0

ClovisWorks Release 2.0

Publication Date:August 2005

Clovis Solutions

Table of Contents

Preface

This manual describes ClovisWorks, an intuitive GUI that simplifies the whole process of custom code generation by providing features that help them to create an Information Model easily.

This Preface describes the following topics:

- Audience
- Document Organization
- Installing ClovisWorks
- Related Manuals
- Getting Help

Audience

The ClovisWorks User Guide is designed for System designers and System deployers of telecom and networking applications. Users of this tool must be familiar with C/C++ programming languages and have prior practical knowledge of working on networking domain. A general overview of UML notations will be an advantage.

Document Organization

The following is a brief description of the remaining chapters of ClovisWorks User Guide:

Chapters	Contents
1 ClovisWorks Overview	Describes the overall functionality, infrastructure of ClovisWorks and provides an introduction to Clovis SISP.
2 Getting Started With ClovisWorks	Explains the various elements of ClovisWorks such as the Clovis Workspace, the Resource Editor, Component Editor, MIB File Explorer, and more.
3 Information Modelling	Describes the steps involved in modeling a Network Element using ClovisWorks GUI. It provides a lucid explanation from Creating a Clovis Project to Modeling a NE to generating customized code for SISP.
4 Component Configuration	Describes the process of configuring the Clovis SISP Components and defining their properties.

Contents - Index - Back

Chapters	Contents
5 Code Generation and Testing	Explains the steps involved in the process of generating SISP codes, compiling, testing, and linking.

Related Manuals

Refer to the following documents for more information about Clovis' products:

Book	Description
CGANs	This is called Clovis Generic Application Notes. Clovis provides a series of CGANs on various topics to help you jump-start on Clovis' SISP components.
SISP API Reference Guide	Describes how to use Application Programming Interfaces (APIs) of each SISP component and provides reference information for each API.

Terminology

There are many related terms in the context of network elements in the telecom world, Clovis Solutions' product offerings and other related terms. In order to effectively present and use these terms some definitions are required. We will apply some terms such as *Resources* and *Components* in the following narration to be able to use each term in a context that is consistent with the goals of this User Guide.

A *Resource* is an entity that can be managed by the Clovis SISP (System infrastructure software platform). A *Resource* may be a hardware equipment (a blade) or a software abstraction implemented by programs running on that hardware (a software process). The Clovis *Resource* definition corresponds to the SAF definition of a resource.

A *Component* in SAF terminology is a software or hardware entity or group of entities on which error reporting and recovery can be performed. There can be different types of *Components* such as a Node, Cluster, Service Unit and so on. In order to flush out ambiguity in this manual the meaning of *Component* is used prudently which does not have any other direct or indirect meaning or reference. This document refers to SISP components such as FM (Fault Manager), AM (Alarm Manager), COR (Clovis Object Registry), as Clovis SISP Components or Clovis Components or SISP Components.

For a complete list of terms on the terminology, refer to CGAN002 - Glossary of Terms.

Contents - Index - Back

References

The ClovisWorks GUI is built on the Eclipse Modeling Framework (EMF), which is a powerful code generation utility for Java applications. As you continue working with ClovisWorks, you will come across a host of Eclipse menu options that may or may not be useful to you. You might want to refer to the Eclipse Online help or Eclipse's website for more information.

- Eclipse 2.1.2 Online Help (www.eclipse.org - online documentation)

Requirements

System Requirements

The following summarizes the system requirements necessary to install and run ClovisWorks.

Hardware Requirements

« 1 GHz and above Pentium processor
« At least 256 MB of memory
« At least 140 MB of disk space

Software Requirements

« Operating System - Windows XP or Linux (Red Hat 9.0)

Installing ClovisWorks

On Windows Platform:

« Unzip the file `clovisworks-SDK-1.0.0-win32<date>.zip`
« In the **<install>/clovisworks** directory, double click the `clovisworks.bat` file to start ClovisWorks.

On Linux Platform:

« Unzip the file: `clovisworks-SDK-1.0.0-linux-<date>.zip`
« Launch ClovisWorks by executing the `./clovisworks.sh` command in the **<install>/clovisworks** directory.

Getting Help

If you need any assistance while working with Clovis products, contact the Clovis Solutions' Customer Application Center (CAC):

- Online: www.clovissolutions.com

Contents - Index - Back

Documentation Feedback

We are interested in hearing from our customers on the documentation provided with this product. Please let us know what you like or do not like about Clovis' technical documentation and let us know if you have any suggestions on improving the documentation. Send your feedback comments to techpubs-feedback@clovissolutions.com.

Chapter 1 ClovisWorks

This chapter covers the following topics:

1.1 ClovisWorks Overview

ClovisWorks is a Graphical User Interface (GUI) tool designed to simplify and accelerate the development of Telecom application software over Clovis' SISP platform. ClovisWorks' graphical interface enables you to capture the Information Model through UML notations and saves them as XML files. Besides providing simple and powerful mechanism with easy drag and drop features to create Resources and Components and define their relationships, ClovisWorks also provides the ability to define attributes of these Resources and Components.

ClovisWorks enables you to rapidly create an information model of the Resources that will be required to manage, and generates customized code for SISP, a flagship product of **Clovis Solutions**. Apart from enabling to easily generate code, it will also provide the capability to compile and deploy the code to the target system under development.

1.2 Features of ClovisWorks

ClovisWorks helps you perform the following:

- Model the hardware, software as well as the application to help manage and administer them.
- Model/design Resources and Components using UML notations to build an Information Model.
- Exchange management information between network devices by providing SNMP MIB (management information base) support.
- Import pre-defined SNMP MIBs.
- Configure Clovis SISP components using a user-friendly GUI.
- Integrate the generated code with Clovis supplied Infrastructure components (SISP) to create SISP for the customer's NE.
- Generate code, build (compile and link) and deploy it to the target system under development with a single click from the UI.
- Represent a number of resources within the information model, which can even scale up or down to incorporate complex information model.

Contents - Index - Back

- Define dependencies and relationships of Resources and represent them as managed objects in a hierarchical structure. These Resources have attributes that represent the actual state of the physical resource.

Other Features

Some of the general features that ClovisWorks provides include:

- Automatic back up of files.
- Enables defining of different roles for the users such as Systems Architect, Systems Developer, Tester to access the project information.
- Options to re-use the pre-defined managed objects across different projects.
- Easy-to-use editors for modeling Resources and configuring the System Infrastructure Software.
- A standard menu interface for managing and building projects.
- Customized views and perspectives.
- Adheres to the analysis and design modeling techniques proposed by the SA Forum, CIM and DMTF to create Resources.

1.3 Developing a System using ClovisWorks

Developing a system using ClovisWorks is a three-fold process. The steps involved in this process are outlined below:

1. Modeling or Designing
2. Coding or Development
3. Building

This three step process is graphically depicted in the following diagram:

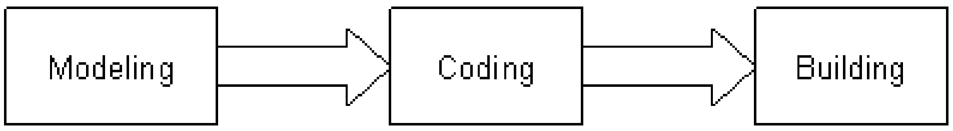


Figure 1-1 Three-fold Process

As described in Figure 1-1 , the first step in the process is modeling the Resources. As part of modeling, you will first define Resources and Components and then define their attributes and characteristics. Next, you will associate Components with the corresponding Resources by defining the relationships. Finally, you will configure the SISP components based on the type of Resources you modeled.

Once you're done with defining the information model using ClovisWorks, you can generate code for your custom information model. ClovisWorks generates a set of C

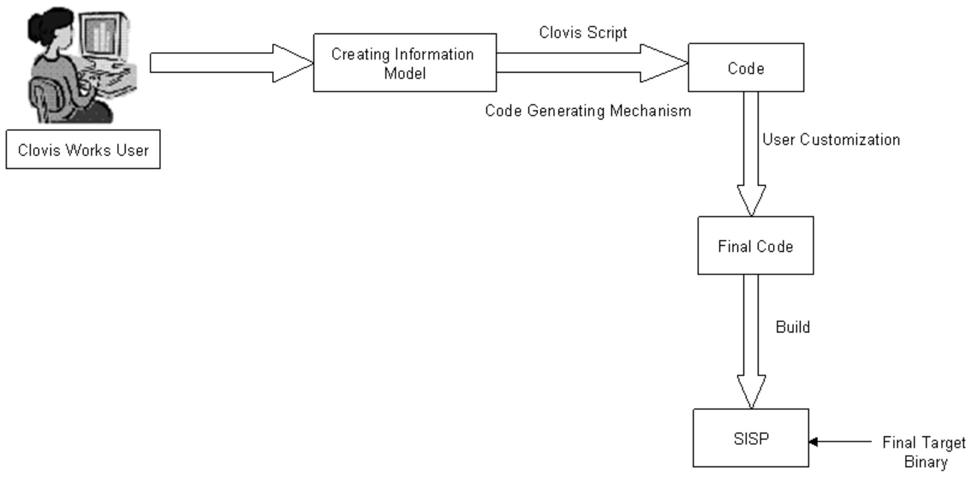
Contents - Index - Back

files and an XML file that a user can modify and add additional code to customize it. You will then compile and link the code to the SISP libraries to build the platform software customized for the target application.

ClovisWorks Flow

The ClovisWorks development framework helps you to easily design, develop, configure, and integrate customized applications onto SISP.

The following figure illustrates how ClovisWorks simplifies the creation of customized application for managing and administering a host of Resources.



ClovisWorks: Flow Diagram

Figure 1-2 Generating SISP-compatible code using ClovisWorks

The Figure 1-2 illustrates at a high level how you can use ClovisWorks interface to model the system information and rapidly generate customized code for deploying onto the SISP.

The System Model (The NE to be modeled) forms the input for ClovisWorks. You can model the system using the intuitive GUI and various features provided by ClovisWorks. Once the various objects of the system to be managed are represented as Resources in ClovisWorks, you need to define a corresponding component and associate the component with the respective resource for adherence to SAF (Service Availability Forum) compliance.

You will then define the build-time and boot-time configuration parameters. All this information is saved automatically in an XML file that will form the input for code generation. Using ClovisWorks, you can generate code for the modeled Network Element. ClovisWorks generates source code and a make file. Using this make file, the system developer can compile and link the code to SISP libraries to make it an executable. This executable is deployed onto the hardware.

Contents - Index - Back

ClovisWorks graphical development environment complements SISP by greatly simplifying the system definition, development, debugging and deployment of SISP-based software. When you create an Information Model you might be required to configure Clovis SISP components based on the type of Resources you might have modeled.

Important!: Configuring Clovis SISP components makes sense only for those resources where SISP is running. Depending upon the business requirement, you can use ClovisWorks to create customized SISP by modeling the system information and configuring the Clovis SISP components associated with the modeled elements.

1.4 Introduction to Clovis SISP

Clovis SISP is a comprehensive suite of software subsystems, libraries, utilities, and tools for system and networking products development. Using SISP, it is possible to build a broad spectrum of products ranging from 1U or 2U types access devices to multi-chassis and cluster-based mission critical carrie-grade platforms. SISP provides product flexibility and modularity for system software to enable system software development through its unique architecture. SISP also helps in enhancing the stability, reliability, and availability of systems by providing an infrastructure that dynamically detects and/or corrects run time hardware and software issues.

SISP is a collection of Libraries (APIs). Each component of SISP like Alarm Manager, Fault Manager and so on, has a collection of APIs that can be used to provide specific functionality to the NEs.

For example, a customer has to define the boot profile and the configuration parameters for the blades and the SISP components. Defining these manually will be a time-consuming process. ClovisWorks greatly simplifies the process of defining the boot profile and the configuration parameters. Using ClovisWorks, you can define the IM for the system and capture attributes that manage the system.

The Clovis SISP architecture comprises various components. They are listed below:

- Communication Core
  - IOC – Intelligent Object Communication
  - EO – Execution Object
  - RMD – Remote Method Dispatch
  - CIDL – Clovis Interface Description Language
  - DEM – Distributed Event Manager
- BIC – Basic Infrastructure Core Library
  - BM – Buffer Management
  - Containers – Various data structures
  - RBE – Rule Based Engine
  - Timer – Multiple soft-timers

  - SM – State Machine
  - HAL – Hardware Abstraction Layer
  - DO – Device Object
- OSAL – Operating System Abstraction Layer
- DBAL – Database Abstraction Layer
- Debug Infrastructure
  - Standalone Debug CLI
  - Distributed Debug CLI
- COR – Clovis Object Registry
- IM – SAF Compliant Information Model
- CPM – Component Manager
- AM – Alarm Client and Server
- PL – Provisioning Library
- FM – Fault Manager
- CM – Chassis Manager based on HPI
- ML – Mediation Library
- SNMP – SNMP sub-agent
- CPS – Checkpoint Service
- Software Exception Handling
- Component Restart

For more information on each Clovis' SISP component, refer to the CGAN series provided with this product.

Contents - Index - Back

1.5 Key Terms

The following key terms must be clear to you before you really start building an Information Model:

- Resources
- Components

1.5.1 Resources

A Resource is an entity that can be managed by the Clovis SISP (System infrastructure software platform). A Resource could be a hardware equipment (a blade) or a software abstraction implemented by programs running on that hardware (a software process). You must represent anything you want to manage as a ‘Resource’ because it has attributes and operations related to manageability of the resource.

1.5.2 Components

A component is a logical entity that represents a set of resources, such as processes. A service unit encapsulates the components representing hardware and/or software resources into a single management unit.

For a complete list of terms, refer to CGAN002 - Glossary of Terms.

Chapter 2
Getting Familiar With ClovisWorks

This chapter describes the following topics:

2.1 ClovisWorks GUI Environment

This section familiarizes you with the ClovisWorks environment. The ClovisWorks GUI is built on the Eclipse Modeling Framework (EMF), which is a powerful code generation utility for Java applications. The primary objective of building ClovisWorks on Eclipse was to leverage the full potential of EMF, which unifies three important technologies Java, UML, and XML. Since ClovisWorks is built onto the Eclipse platform, you come across a host of menu and toolbar options that are commonly inherent to Eclipse Platform and yet are useful to the ClovisWorks users. Look up eclipse help for detailed information on using menu and toolbar options of eclipse.

The Following are the main elements of the ClovisWorks GUI:

- Clovis Workspace
- IDL Definition
- Boot Configurations
- Build Configurations
- Add Alarm Profiles
- Resource Editor
- Component Editor
- Node Profile
- Update Script and Build Files
- Outline View
- MIBTree View

Contents - Index - Back

2.2 Clovis Workspace

The Clovis Workspace lists all the ClovisWorks projects in a hierarchical structure. It lists down all the corresponding files related to the project in a tree-like structure, such as source files, configuration files, and Clovis scripts.

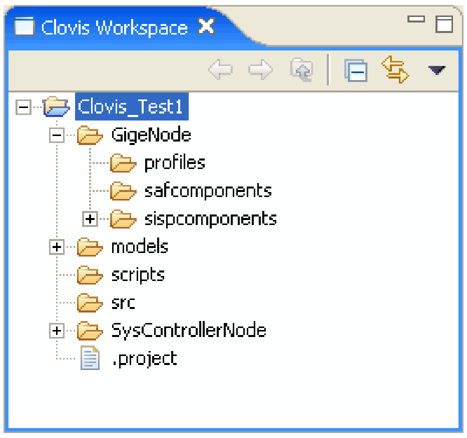


Figure 2-1 Clovis Workspace

Figure 2-1 displays the Clovis Workspace that lists the project and its associated files and directories. In the above case, Clovis_Test1 is the name of the project that has a few directories and associated files. You can open and close projects in the Clovis Workspace.

Following are the default directories available under each project directory.

- **models** —Contains files related to the NE information model.
- **scripts** —Contains the script files.
- **src** —Contains the source files.
- .**project**—Contains the project file.

You can also filter the project directories based on your need. The next section details you about filtering project directories.

Contents - Index - Back

2.2.1 Filtering Project Directories

You can choose to hide or display the default directories under each project. To hide or display directories, click the drop-down arrow on the Clovis Workspace titlebar. Select *Filters* from the menu that appears, as shown in the Figure 2-2 .

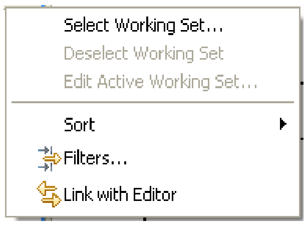
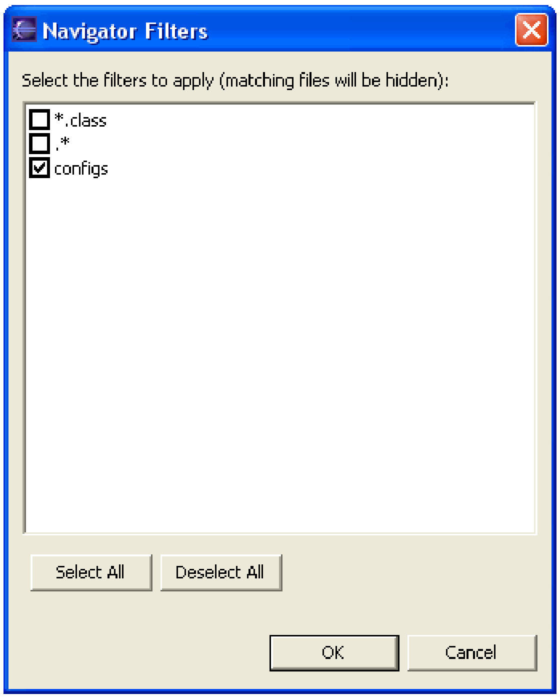


Figure 2-2 Filters Option

The Navigation Filters dialog appears.



Select the filters to apply (matching files will be hidden):

☐ *.class
☐ .*
☑ configs

Figure 2-3 Navigator Filter

Each check-box in the *Navigator Filters* dialog shown in Figure 2-3 corresponds to the default directories to be displayed under the target project. Selecting any of the check boxes will hide the corresponding directory under the project in the Clovis Workspace. To display the directory again, clear the respective directory check-box. You can also click **Select All**, to select all the directories or files listed or **De-select All** to clear all the directories and files listed from being filtered. Click **OK** to apply the changes.

2.2.2 Using Context Menu in Clovis Workspace

The Workspace provides a context sensitive right-click menu using which you can perform various tasks. The context menu is shown in Figure 2-4 and its contents are explained in the following Table. The context-menu shown below has various options that are part of Eclipse. The Clovis' options are available at the bottom as a cascading menu option. You will be extensively using Clovis' options while at the same time utilizing the Eclipse options that are available in the Figure 2-4 .

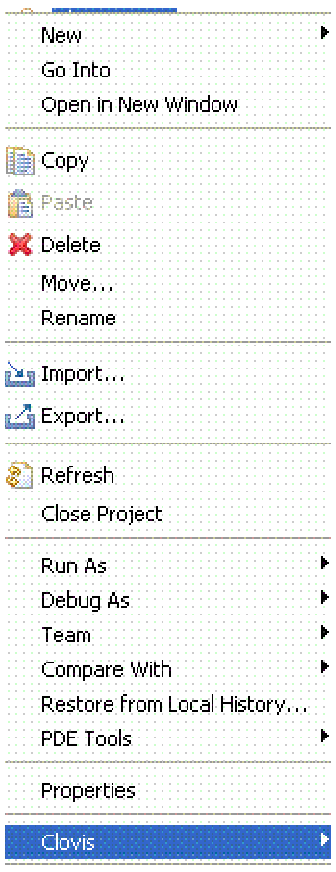


Figure 2-4 Clovis Workspace Menu

Menu Function	Description
New> Project	Opens a new project wizard that will let you create a new Clovis project.
Go Into	Goes to the next level of the project hierarchy.
Open in New Window	Opens the selected resource in a new window.
Copy	Places a copy of the selection on the clipboard
Paste	Inserts the resources from the clipboard to the Clovis Workspace.
Delete	Removes the current selection from the Workspace.

Menu Function	Description
Move	Moves resources from one Workbench location to another (for example, from one project to another project).
Rename	Renames the selected resource.
Import	Launches a wizard that allows you to import the following into the Clovis Workspace: • Existing Project into Workspace • External Features • External Plug-ins and Fragments • File system • Team Project set • Zip file
Export	Launches a wizard that allows you to export the following from the Clovis Workspace: Deployable features Deployable plug-ins and fragments File system JAR file Java doc Team Project Set Zip file
Refresh	Refreshes the contents of the Clovis Workspace.
Close Project	Closes a project.
Run	Re-launches the most recently used application.
Debug	Re-launches the most recently used application under debugger control.
Team	Not supported in ClovisWorks Release 2.0
Compare With	Not supported in ClovisWorks Release 2.0
Restore From Local History	Restores the recently deleted resource(s) from the local history.
Properties	Displays a window showing all the property details of the current selection, such as—path, type, location and last modified.
Clovis	This cascading menu has various options which are inherent to ClovisWorks. It is explained in Clovis Menu.

To close the Clovis Workspace window, click “x”. You can re-open the Workspace by clicking ***Window*** > ***Show View*** > ***Other*** and then selecting Clovis Workspace. This is illustrated in Figure 2-5 .

Contents - Index - Back

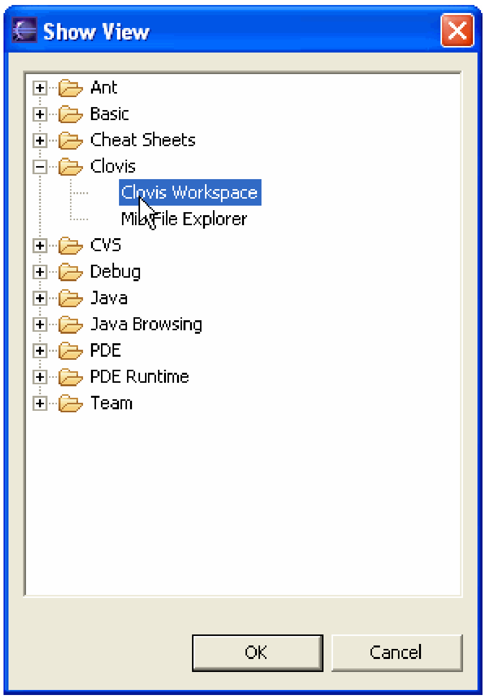


Figure 2-5 Show View dialog

2.2.3 Clovis Menu

You can perform various Clovis' project-related tasks mainly from the Clovis' Sub Menu, shown in Figure 2-6 . It provides 7 options that you can access by right-clicking in the Clovis Workspace, choosing Clovis and then the respective option.

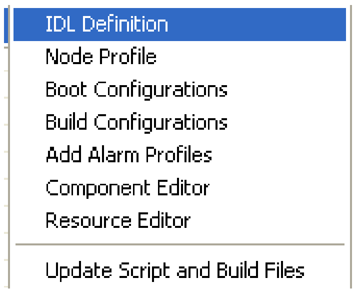


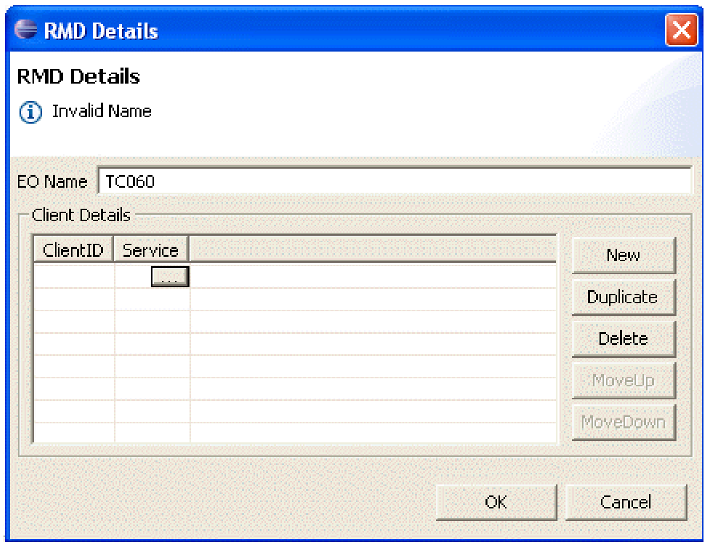
Figure 2-6 Clovis Sub Menu

IDL Definition	Enables you to define EO interface and related data structures.
Boot Configurations	Enables you to configure SISP components during boot time.
Build Configurations	Enables you to configure SISP components during build time.
Add Alarm Profiles	Enables you to add Alarm profiles

Component Editor	Opens the editor in the middle pane. Offers you to create Components and define their properties.
Resource Editor	Opens the editor in the middle pane. Offers you to create Resources and define their properties.
Node Profile	Enables you to specify node configurations.
Update Script and Build Files	Enables you to sync up to the latest scripts available with the product.

2.3 Specifying IDL Information

You use the IDL interface dialog to define the EO interface and related data structures. To launch IDL dialog, right click and select IDL Definition from the Clovis submenu. The RMD details dialog appears, as shown in


RMD Details

RMD Details

Invalid Name

EO Name: TC060

Figure 2-7 IDL RMD Details

EO NAME	Name of the EO
ClientID	Client Id of the EO's client table
Service	Service Details. Click [...] to specify Service details, as shown in Figure 2-8 .

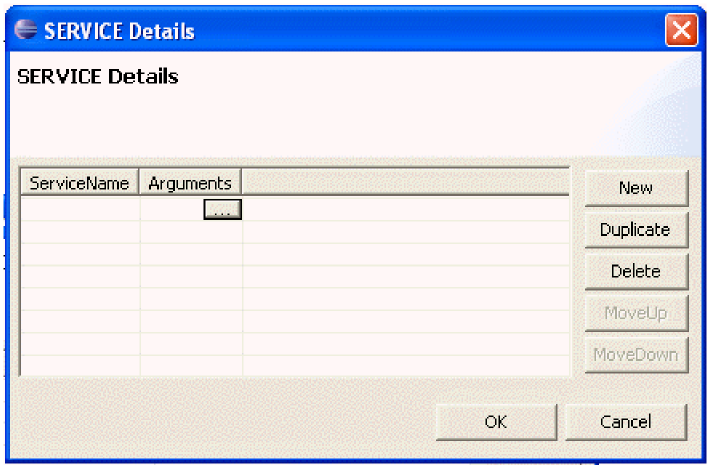
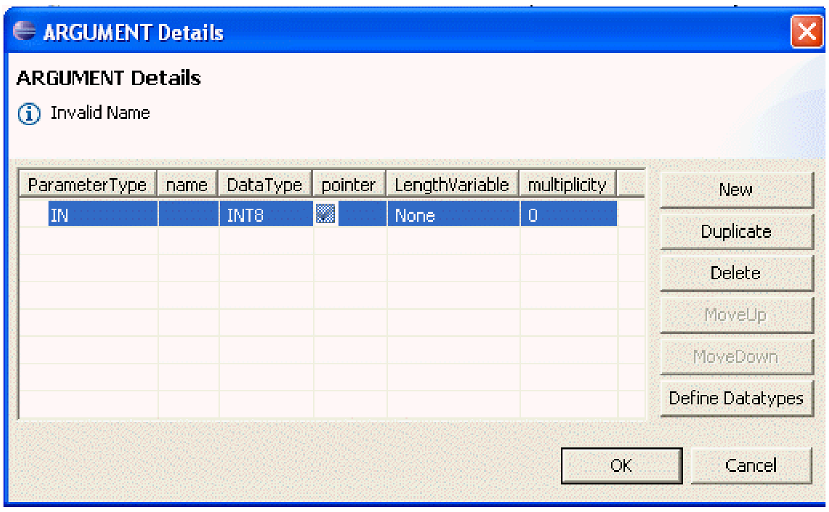


Figure 2-8 IDL Service Details

Service Name	Service name of the client
Arguments	Argument Details. Click ... to specify Argument details as shown in Figure 2-9 .



ParameterType	name	DataType	pointer	LengthVariable	multiplicity
IN		INT8	☐	None	0

Figure 2-9 IDL Argument Details

Parameter Type	Type of the parameter
Name	Name of the argument

DataType	Data type
Pointer	Whether the argument is of pointer type
LengthVariable	Length of the variable

After specifying the Argument details, you need to define the data types by clicking the Datatypes button on the Argument details screen. The IDL Spec details screen appears, as shown in Figure 2-10 .

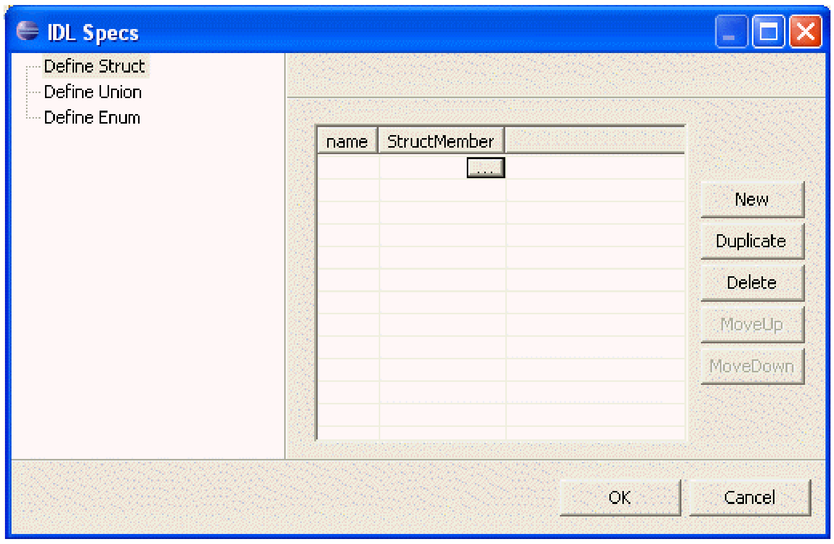


Figure 2-10 IDL DataTypes

In the above screen you can define the Structures, Unions, and Enums. The following section describes the details of defining the Structs, Unions, and Enums.

Defining Structs

To define the Structs, Select it in Fig - 10, specify the name and click ... to launch DataMember details dialog as shown in Figure 2-11 .

Contents - Index - Back

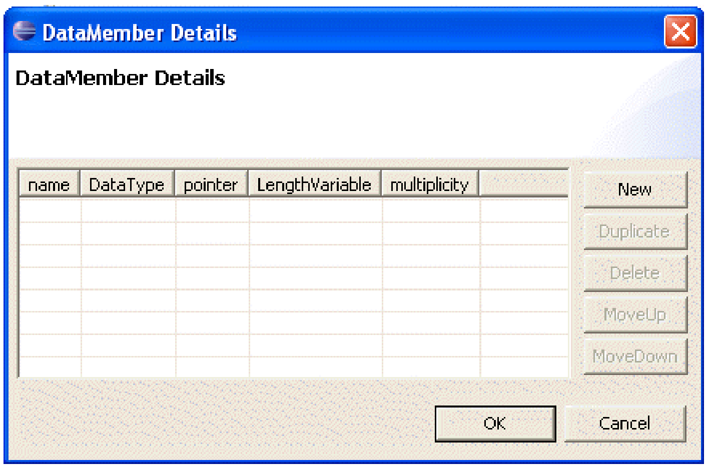


Figure 2-11 Data Member Details

Name	Name of the data member
DataType	Data type
Pointer	Whether it is of pointer type
LengthVariable	Length of the variable

Defining Unions

To define the Unions, Select it in Fig - 10, specify the name and click [...] to launch DataMember details dialog as shown in Figure 2-11 . Follow the process as explained for Defining Structs in the previous section.

Defining Enums

To define the Enums, Select it in Fig - 10, specify the name and click [...] to launch EnumMember details dialog as shown in Figure 2-12 .

Contents - Index - Back

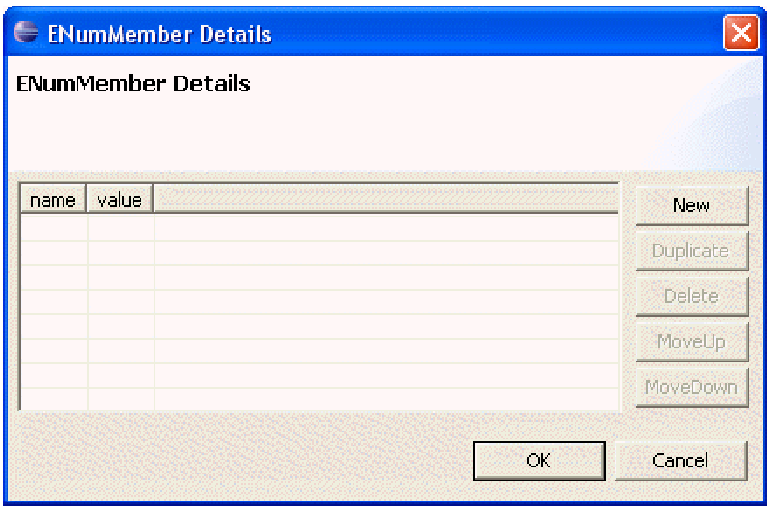


Figure 2-12 EnumMember Details

Specify the name and value and click OK.

2.4 Using Boot Configuration

You can launch the Boot Configuration dialog from the Clovis Workspace menu. To launch the boot configuration dialog, right click and select Boot Configurations from the Clovis submenu. The boot configuration dialog appears, as shown in Figure 2-13 .

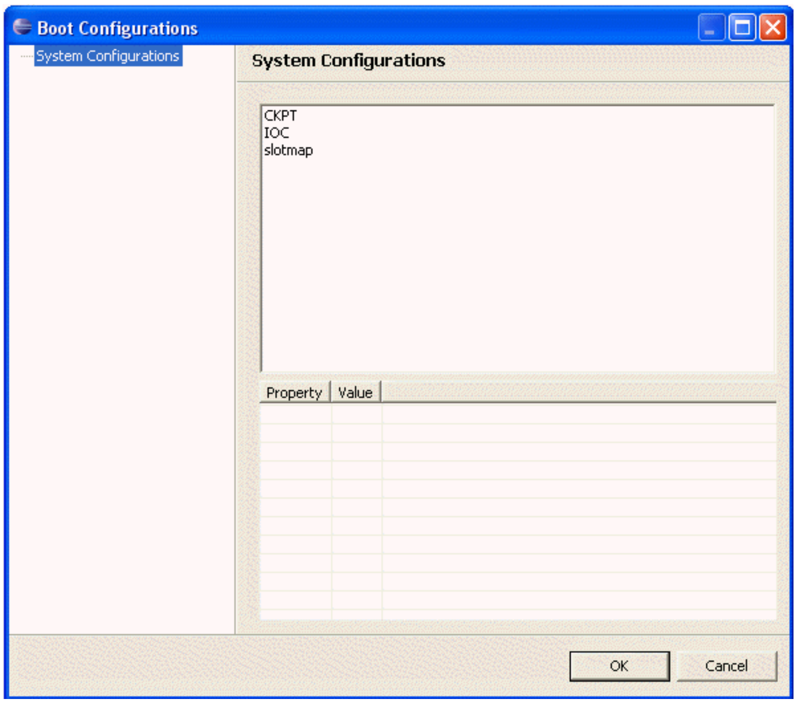


Figure 2-13 Boot Configuration Dialog

Contents - Index - Back

You can configure the properties of the respective SISP components during boot time in the above figure. For more details on configuring each property for the respective components, refer to Chapter 4Configuring Components.

2.5 Using Build Configuration

You can launch the Build Configuration dialog from the Clovis Workspace menu. To launch the Build Configuration dialog, right click and select Build Configurations from the Clovis submenu. The Build configuration dialog appears, as shown in Figure 2-14 .

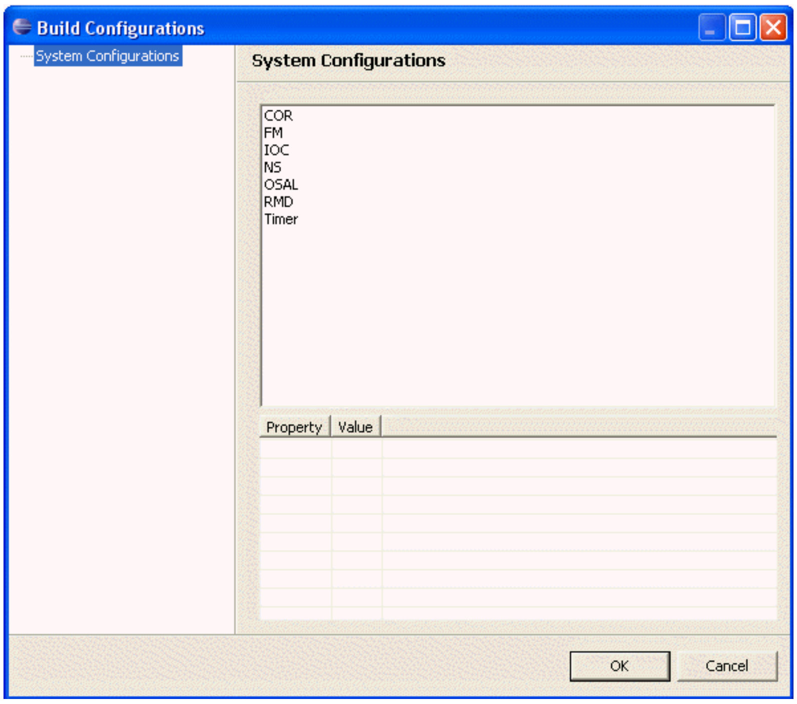
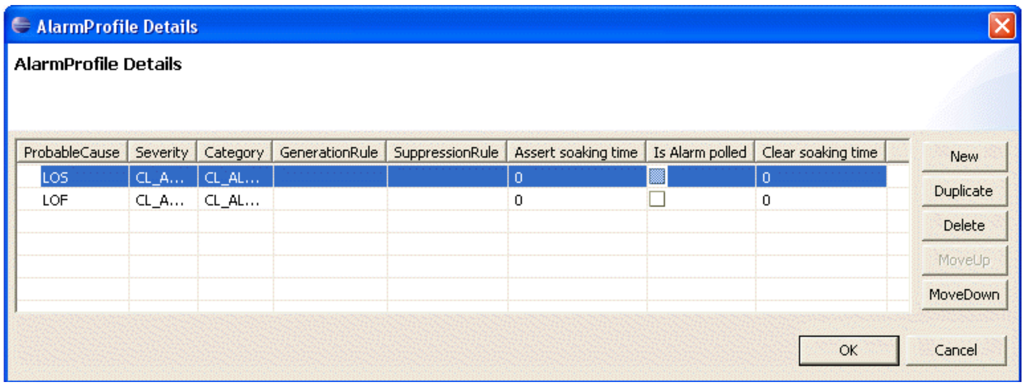


Figure 2-14 Build Configuration Dialog

You can configure the properties of the respective SISP components during build time using the above dialog. For more details on configuring each property for the respective SISP components, refer to Chapter 4Configuring Components.

2.6 Defining Alarm Profiles

You can add Alarm profiles using the option from the Clovis Workspace menu. To launch the Alarm Profiles dialog, right click the Clovis Workspace menu and select Add Alarm Profiles. The Add Alarm Profiles dialog appears, as shown in Figure 2-15 .

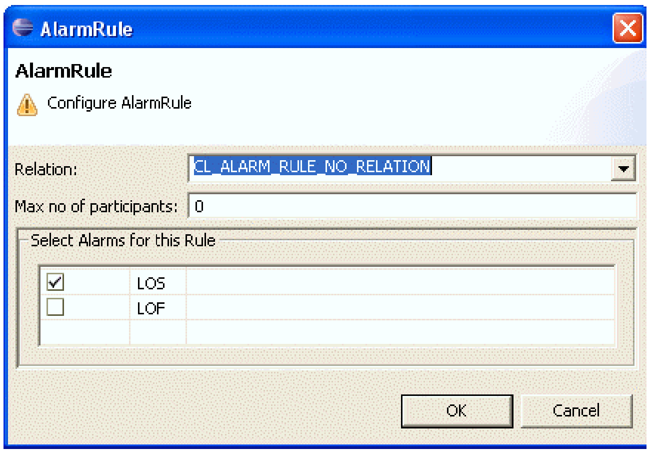


ProbableCause	Severity	Category	GenerationRule	SuppressionRule	Assert soaking time	Is Alarm polled	Clear soaking time
LOS	CL_A...	CL_AL...			0	▨	0
LOF	CL_A...	CL_AL...			0	☐	0

Figure 2-15 Alarm Profiles Dialog

	Description
Probable Cause	Probable cause of the alarm.
Severity	Defines the severity of the alarm.
Category	Defines the category of the alarm.
GenerationRule	Indicates the alarm(s) which must be present for this alarm to be generated.
SuppressionRule	The suppression rule specifies the alarms that can suppress this alarm.
AssertSoakingTime	Defines the assert soaking time.
IsAlarmPolled	If alarm is polled or not.
ClearSoakingTime	Defines the clear soaking time of alarm.

To add an Alarm rule, click ... in the GenerationRule field. The Alarm rule dialog appears, as shown in Figure 2-16 .



Relation: CL_ALARM_RULE_NO_RELATION
Max no of participants: 0
Select Alarms for this Rule
☑ LOS
☐ LOF

Figure 2-16 Alarm Rule Dialog

Relation	Specify the relation of the alarm.
Maximum number of participants	Specify the maximum number of participants
Select Alarms for this Rule	Select the desired check-box to select the respective alarm. For example, if you want LOS (Loss of Signal) alarm, select the corresponding check-box.

To suppress a rule, click ... in the SuppressionRule field. The Suppression rule dialog appears, as shown in Figure 2-16 . Specify the Suppression rules and click OK.

2.7 Using Resource Editor

Once you are done with creating a project, you will start defining the Information Model in the Resource Editor. Resource Editor lets you easily create all the resources you require to manage through its graphical interface with the help of UML notations. The Resource Editor's simple yet powerful mechanism with easy drag and drop features also let you define the properties of the resources. Besides defining the properties and attributes of the resource, you can also specify the attributes and operations of Provisioning, Chassis Management and so on.

The Resource Editor supports UML containment, inheritance, and composition. It provides a set of default, tailor-made meta classes in a Palette for your instant use. You can drag and drop them from the palette to the editor area to create class types and then use the relation connectors to define the relationships between them.

To open the Resource Editor, right click the mouse from the ClovisWorkspace and select Resource Editor. The Resource Editor displays, as shown in Figure 2-17

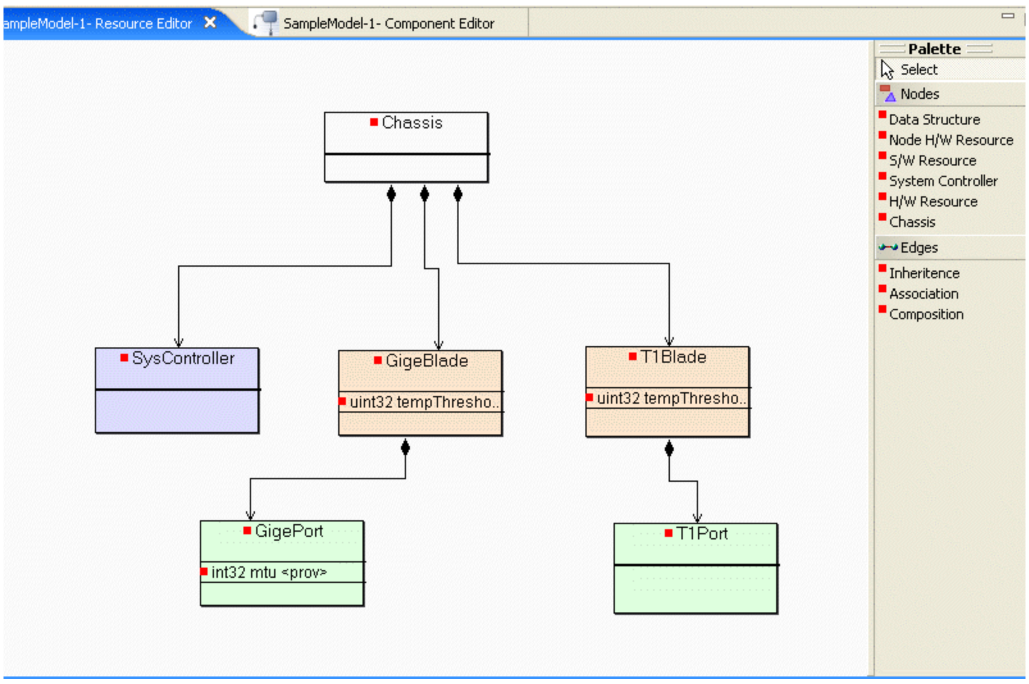


Figure 2-17 Resource Editor

In the Resource Editor illustration above, you can see the Resource Editor Area and the Palette, which have been appropriately labeled. On the right hand side is the palette that contains the six default meta classes.

- Data Structure
- Node Hardware Resource
- Software Resource
- System Controller
- Hardware Resource
- Chassis

The ClovisWorks Resource Editor offers easy-to-use features that let you create the class type you want and define the relationship between them using the relation connectors. To define the information model, you must drag and drop the required meta class from the Palette Area to the Resource Editor Area. Each pre-defined meta class is represented by a unique color. The pre-defined meta classes are available in a palette, as shown in Figure 2-18

Contents - Index - Back

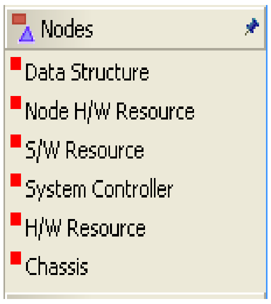


Figure 2-18 Meta Classes Palette

The next section describes each meta class in detail and explains how to use them.

2.7.1 Data Structure

Select this predefined meta class from the meta classes palette to create a user-defined data structure.

Figure 2-19 Data Structure

The Data Structure meta class is represented by grey color. The above figure displays the Data Structure meta class in the palette. To model a Data Structure meta class in the Resource Editor, perform the following steps:

1. Click Data Structure in the Palette.
2. Drag the cursor onto the Resource Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create the Data Structure meta class model in ClovisWorks.
3. Drag and drop the object in the Resource Editor. The object is available in the Editor. A sample of the object looks as follows:

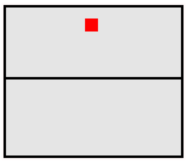


Figure 2-20 Data Structure Representation

You can define the properties of Data Structure by double clicking the above object. Double click the object. The properties dialog pops up, as shown in Figure 2-21 :

Contents - Index - Back

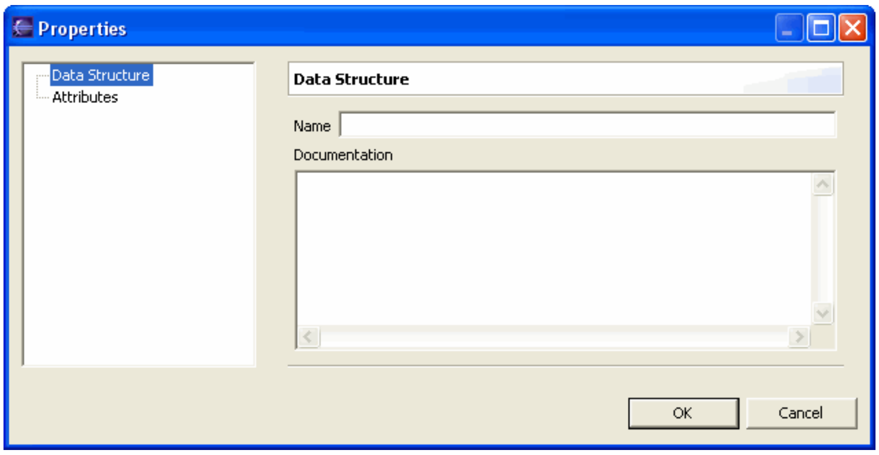


Figure 2-21 Data Structure Properties Dialog

Select Data Structure in the above dialog. The Data Structure screen displays. Specify a name for the data structure in the Name field. Use the Documentation text box to capture the relationship or some description of the data structure.

To define the attributes, select Attributes in the above dialog. The dialog box changes, as shown in Figure 2-22 . Define all the attributes as you require.

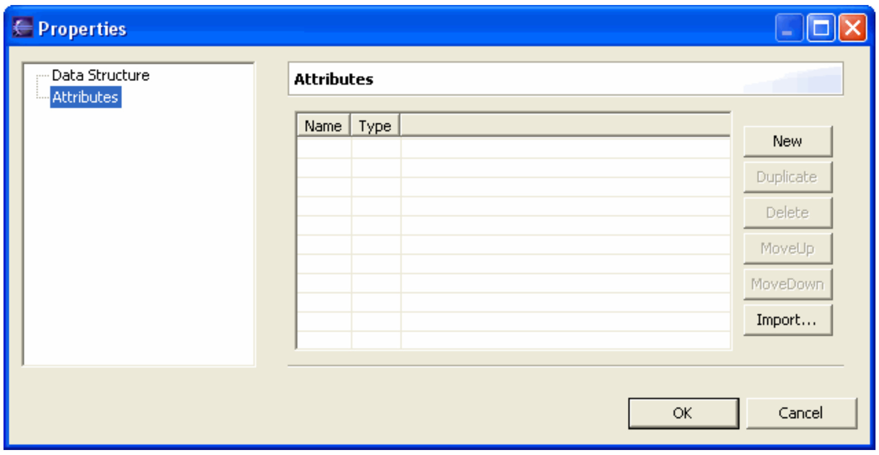


Figure 2-22 Data Structure Attributes Dialog

Click OK once you're done.

The following table lists the options and explains them.

Function	Description
New	Adds new attributes. Attributes could be of the types: int, long, short, char, and array.
Duplicate	Creates a duplicate of the selected attribute.
Delete	Deletes a selected attribute.
Move Up	Moves the selected attribute up the index.
Move Down	Moves the selected attribute down the index.
Import	Imports attributes from MIBs. The detailed procedure to import from a MIB is explained in the next section Importing from a MIB.

Note: You can define the attributes either manually (as explained above) or if you have the attributes pre-defined in a MIB (management information base) you can load them. The next section explains the procedure of importing MIBs.

Importing from a MIB

ClovisWorks provides an option to import from standard MIBs. If you have your attributes defined, you can easily import them using the MIB import facility. To import from a MIB, click Import from a MIB in the Data Structures Attributes dialog. The MIB Attributes Selection dialog appears, as shown in Figure 2-23 .

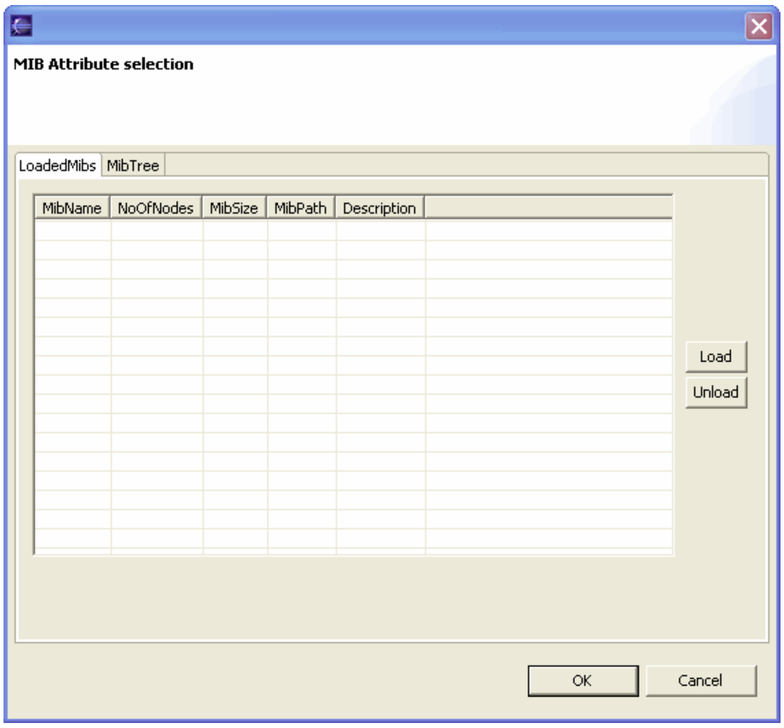


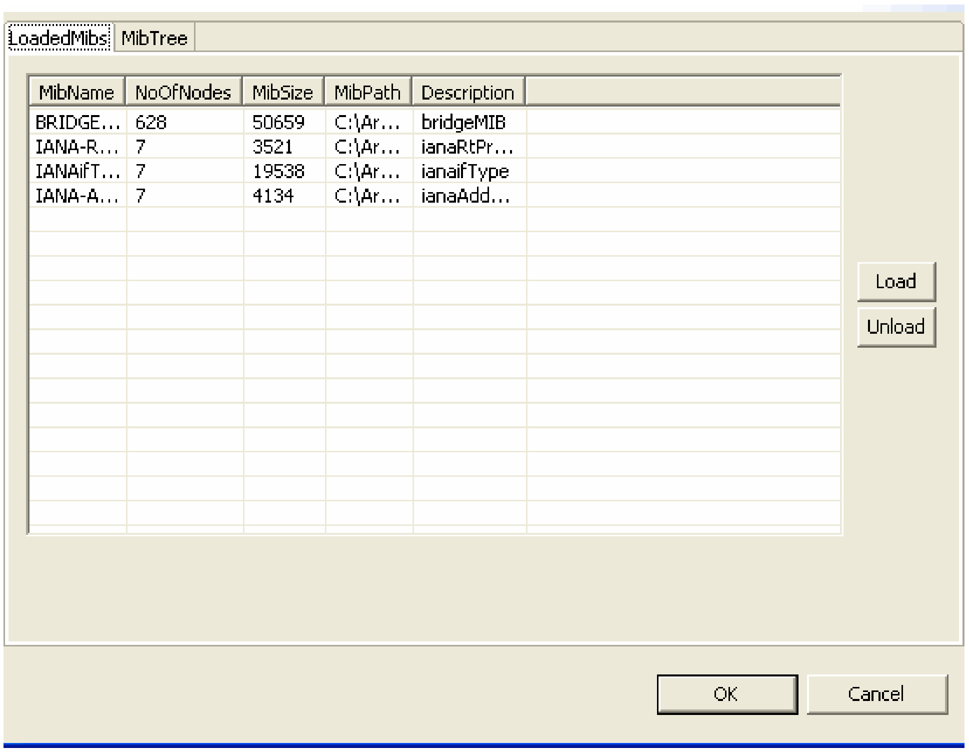
Figure 2-23 MIB Import Dialog

Contents - Index - Back

There are two tabs —Loaded MIBs and MIBTree. Use the Loaded MIBs tab to load or unload a MIB.

Loading MIBs:

Click Load to load a MIB.The 'Open' dialog appears. Navigate to the location where you have MIBs stored, select and click Open to load the MIB. For example, when you load a MIB, the MIB import dialog appears as shown in Figure 2-24 .



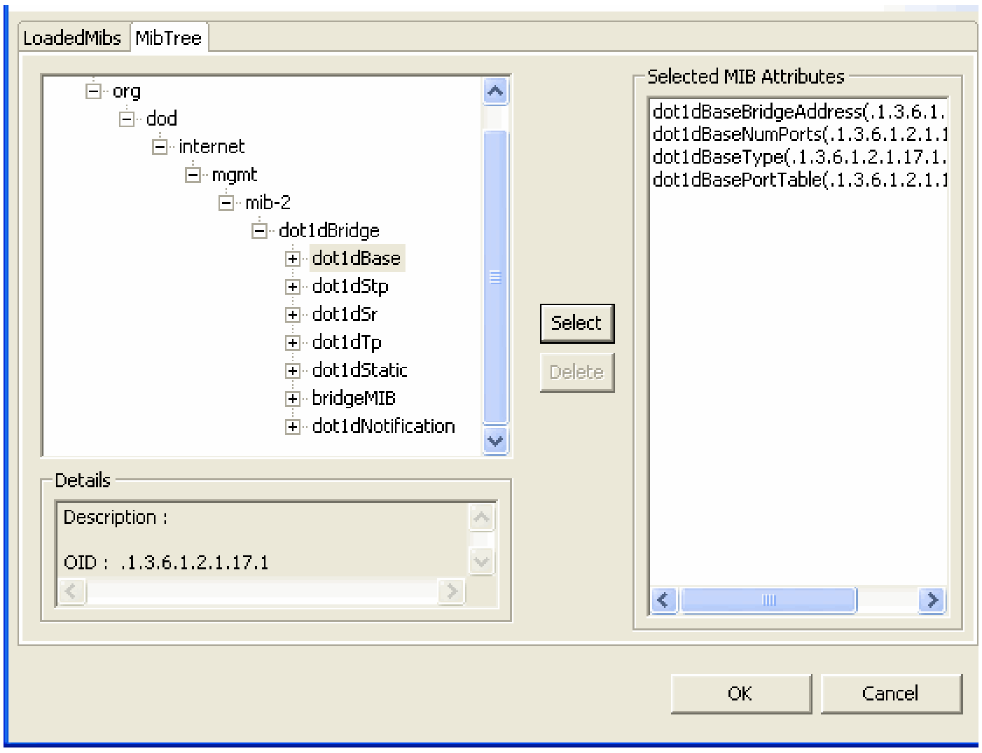
MibName	NoOfNodes	MibSize	MibPath	Description
BRIDGE...	628	50659	C:\Ar...	bridgeMIB
IANA-R...	7	3521	C:\Ar...	ianaRtPr...
IANAifT...	7	19538	C:\Ar...	ianaifType
IANA-A...	7	4134	C:\Ar...	ianaAdd...

Figure 2-24 Loaded MIBs View

Click the MIBTree tab to see the properties and attributes of the MIBs. In the MIBTree view, you will see a tree view that lists the attributes of the MIBs. This is illustrated in Figure 2-25 .

Unloading MIB

To unload a MIB, select the respective MIB and click Unload. The MIB is unloaded.



Selected MIB Attributes

```
dot1dBaseBridgeAddress(.1.3.6.1.
dot1dBaseNumPorts(.1.3.6.1.2.1.1
dot1dBaseType(.1.3.6.1.2.1.17.1.
dot1dBasePortTable(.1.3.6.1.2.1.1
```

Details

Description :

OID : .1.3.6.1.2.1.17.1

Figure 2-25 MIB Tree View

The MIB tree view displays the details of the MIBs and attributes of the selected MIB. To view the attributes of a MIB, select the MIB in the left pane of MIBTree view and then click Select to display all the attributes of the MIB in the Selected MIB Attributes.

2.7.2 Node Hardware Resource

Select this meta class only when you want to model, say for example, a blade. This meta class is represented by orange color. The Figure 2-18 displays the Meta Class Palette. To model a Node Hardware Resource in the Resource Editor, perform the following steps:

1. Click Node Hardware Resource in the Palette.
2. Drag the cursor onto the Resource Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create the Node Hardware Resource class model in ClovisWorks.
3. Drag and drop the object in the Resource Editor. The object is available in the Editor. A sample of the object appears as shown in Figure 2-26 .

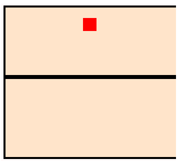
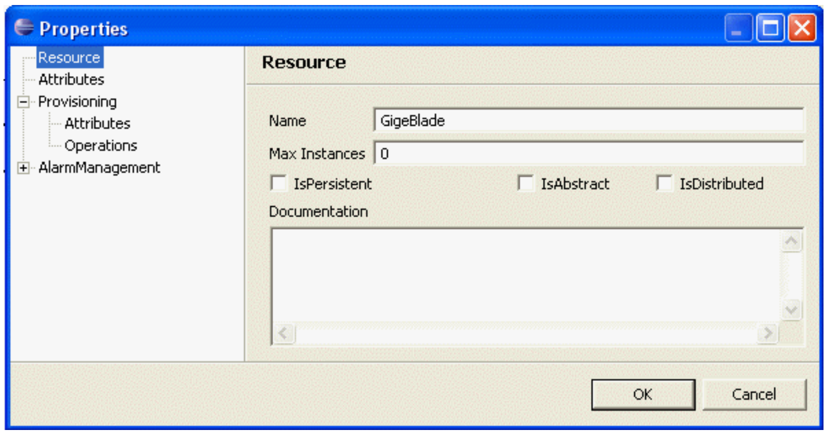


Figure 2-26 Node Hardware Resource Representation

You can define the properties of Node Hardware Resource by double clicking the above object. Double click the object. The properties dialog pops up, as shown in Figure 2-27 .



Name: GigeBlade
Max Instances: 0
☐ IsPersistent ☐ IsAbstract ☐ IsDistributed
Documentation

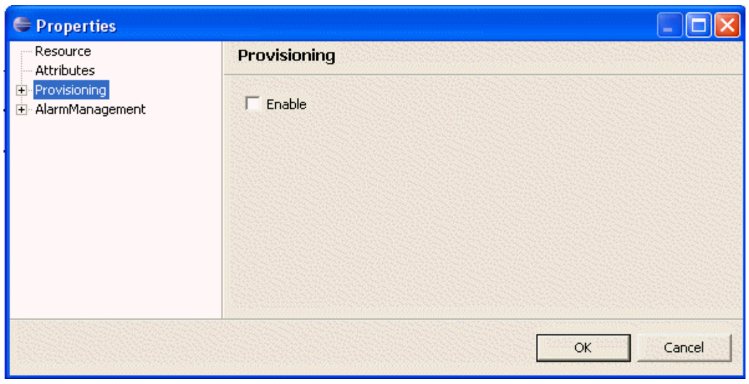
Figure 2-27 Node HW Properties Dialog

Select Resource in the above dialog. The Resource screen appears. Specify a name for the Node Hardware Resource in the Name field.

IsPersistent—Check to capture the persistence behavior of the Resource class.
IsAbstract—Check to capture the abstract behavior of the Resource class.
IsDistributed—This is currently not supported.
Documentation—Use it to capture resource class relation description.

To define the attributes, select Attributes in the above dialog and repeat the process as explained in the section Data Structure.

Click Provisioning. The Provisioning dialog appears, as shown in Figure 2-28 .



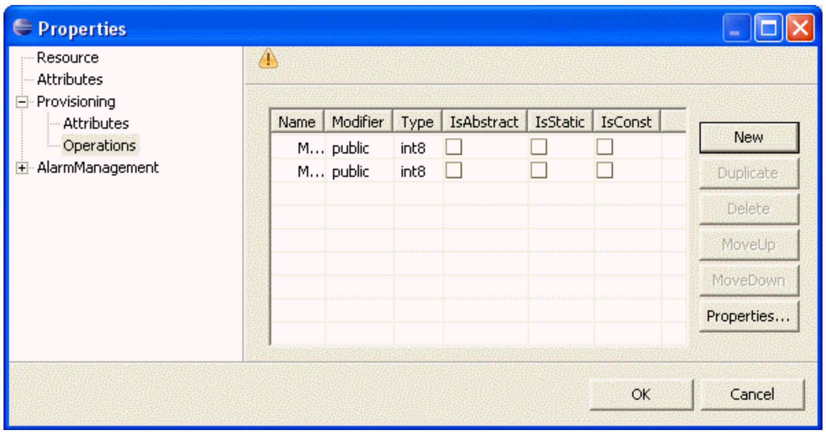
☐ Enable

Figure 2-28 Provisioning Dialog

Select the Enable check-box depending upon your Provisioning requirements. Click the '+' symbol next to the Provisioning dialog. You will see the Attributes and Operations of Provisioning.

You can define the attributes and operations of Provisioning. To define the attributes, select the respective option in the Properties dialog and define them as described in the section Data Structure.

To define the operations, select the respective options and define the operations as needed. The Operations dialog appears upon selecting Operations. The Operations dialog appears as shown in Figure 2-29 .



Name	Modifier	Type	IsAbstract	IsStatic	IsConst
M...	public	int8	☐	☐	☐
M...	public	int8	☐	☐	☐

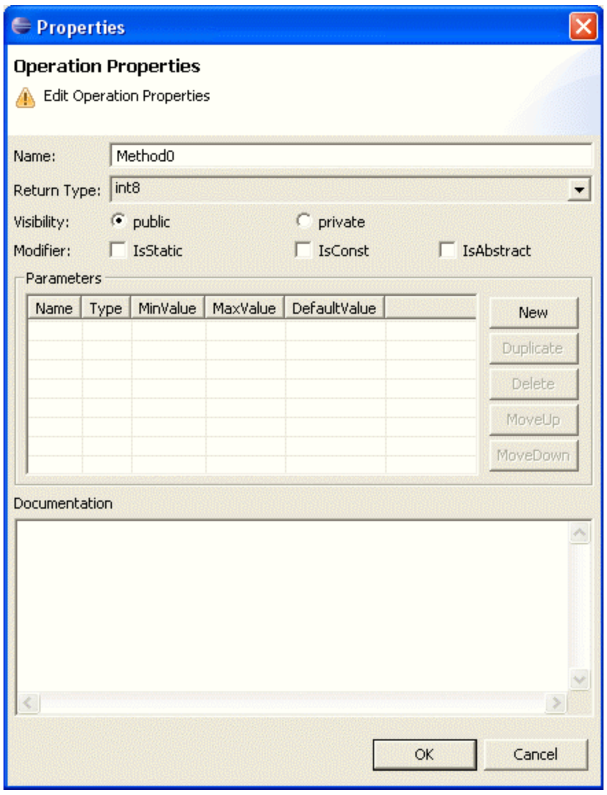
Figure 2-29 Node Hardware Operations Dialog

	Description
Name	Name of the method
Modifier	Modifier type, either public or private.
Type	Return type
IsAbstract	Whether method is abstract
IsStatic	Whether method is static
IsConstant	Whether method is constant

Once you are done with defining the above fields, click anywhere in the row to select it. The Properties button is enabled. Click the Properties button. The Operation Properties dialog appears, as shown in Figure 2-30 . Define the Operation properties as described in the Data Structure.

Note: To invoke the Operation Properties dialog, it is important for you to select one of the fields in Figure 2-29 to enable the Properties button.

Contents - Index - Back


Properties

Operation Properties

Edit Operation Properties

Name: Method0

Return Type: int8

Visibility: [✓ public] private

Modifier: IsStatic IsConst IsAbstract

Parameters

Name	Type	MinValue	MaxValue	DefaultValue

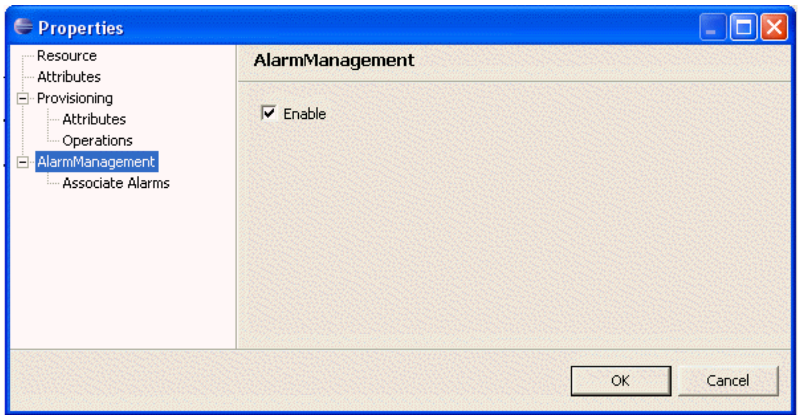
New Duplicate Delete MoveUp MoveDown

Documentation

OK Cancel

Figure 2-30 Operation Properties Dialog

Once you are done with defining the above properties, click the '+' symbol next to Alarm Management. The Alarm Management enable screen appears, as shown in Figure 2-31 .


Properties

- Resource
- Attributes
- Provisioning
  - Attributes
  - Operations
- AlarmManagement
  - Associate Alarms

AlarmManagement

[✓] Enable

OK Cancel

Figure 2-31 Alarm Management Dialog

To define Alarm Management properties, select the enable checkbox to activate Alarm properties. Now, select Associate Alarms to display the Associate Alarm dialog where you can associate or disassociate Alarms. The Associate Alarm dialog is as shown in Figure 2-32 .

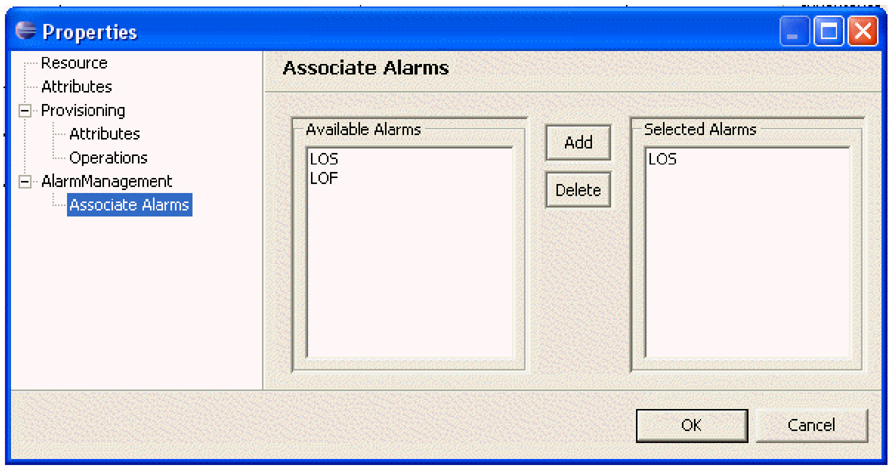


Figure 2-32 Associate Alarms

To add an alarm, select it from the Available Alarms pane and click Add. The selected alarm appears in the Selected Alarms pane. To delete, select the alarm and click Delete.

2.7.3 Software Resource

Select this meta class only when you want to model a software resource. This meta class is represented by pink color. The Figure 2-18 displays the meta class palette. To model a Software Resource in the Resource Editor, perform the following steps:

1. Click Software Resource in the Palette.
2. Drag the cursor onto the Resource Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create the Software Resource class model in ClovisWorks.
3. Drag and drop the object in the Resource Editor. The object is available in the Editor. A sample of the object appears as shown in Figure 2-33 .

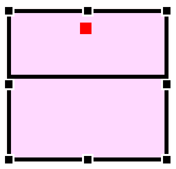


Figure 2-33 Software Resource Representation

You can define the properties of Software Resource by double clicking the above object. The process for defining the attributes and operations for a Software Resource is the same as that of a Node Hardware Resource. Refer to the Node Hardware Resource section for more information on defining the attributes and operations.

Contents - Index - Back

2.7.4 System Controller

Select this meta class when you want to model a System Controller hardware. This meta class is represented by violet color. The Figure 2-18 displays the meta class palette. To model a System Controller in the Resource Editor, perform the following steps:

1. Click System Controller in the Palette.
2. Drag the cursor onto the Resource Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create the System Controller class model in ClovisWorks.
3. Drag and drop the object in the Resource Editor. The object is available in the Editor. A sample of the object appears as shown in Figure 2-34 .

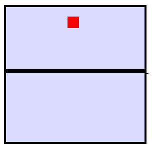


Figure 2-34 System Controller Representation

You can define the properties of System Controller by double clicking the above object. The process for defining the attributes and operations for a Hardware Resource is the same as that of a Node Hardware Resource. Refer to the Node Hardware Resource section for more information on defining the attributes and operations.

2.7.5 Hardware Resource

Select this meta class when you want to model hardware resource. This meta class is represented by green color. The Figure 2-18 displays the meta class palette. To model a Hardware Resource in the Resource Editor, perform the following steps:

1. Click Hardware Resource in the Palette.
2. Drag the cursor onto the Resource Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create the Hardware Resource class model in ClovisWorks.
3. Drag and drop the object in the Resource Editor. The object is available in the Editor. A sample of the object appears as shown in Figure 2-35 .

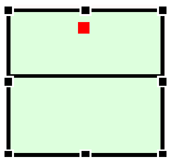


Figure 2-35 Hardware Resource Representation

You can define the properties of a hardware resource by double clicking the above object. The process for defining the attributes and operations for a hardware resource is the same as that of a Node Hardware Resource. Refer to the Node Hardware Resource section for more information on defining the attributes and operations.

Contents - Index - Back

2.7.6 Chassis

Select this meta class when you want to model a Chassis. This meta class is represented by pink color. The Figure 2-18 displays the meta class palette. To model a Chassis Resource in the Resource Editor, perform the following steps:

1. Click Chassis Resource in the Palette.
2. Drag the cursor onto the Resource Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create the Chassis class model in ClovisWorks.
3. Drag and drop the object in the Resource Editor. The object is available in the Editor. A sample of the object appears as shown in Figure 2-36 .

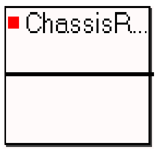


Figure 2-36 Chassis Resource Representation

You can define the properties of a Chassis resource by double clicking the above object. The process for defining the attributes and operations for a Chassis resource is the same as that of a Node Hardware Resource. Refer to the Node Hardware Resource section for more information on defining the attributes and operations.

Contents - Index - Back

2.7.7 Using Edges

Edges provides you with two options— Inheritance and Composition. This is illustrated in Figure 2-37 .

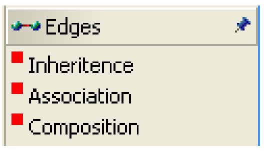


Figure 2-37 Edges Palette

The following section describes Inheritance and Composition in detail:

Inheritance

Inheritance links generalized classes to their specializations. While modelling, you can define the attributes common to a resource using the options in the ClovisWorks GUI and inherit these properties to another resource without actually going through the grind of defining it from the ground up.

To create inheritance relationship between resources, click **Inheritance** from the Edges group in the Palette to select it. The arrow changes to contain an additional circle along with the arrow itself. Now, drag and drop from one resource ‘X’ to another resource ‘Y’ to inherit the pre-defined properties of X to Y.

Association

Association connects objects of one class to objects of another class. If you define association between two classes, you can navigate between an object of one class to an object of another class.
To create Association relationship between classes, select the **Association** option in the *Edges* group and drag between the two classes you wish to associate in the Resource Editor.

Composition

ClovisWorks also provides you with options to define the containment hierarchy from the Edges group. After creating the resources, you must define the containment relationship between some resources. For example, if you have modeled a chassis that contains two blades, then you must use the composition option from the Edges group and define a containment relationship between Chassis and the blades because a Chassis always contains blades. To do this, click Composition option, and then select the modeled Chassis in the Resource Editor Area. From there drag the arrow to the blade and click the mouse one time and leave. The containment relationship is defined. The containment relationship between the Chassis resource and Node Hardware resource will appear as shown in Figure 2-38 .

Contents - Index - Back

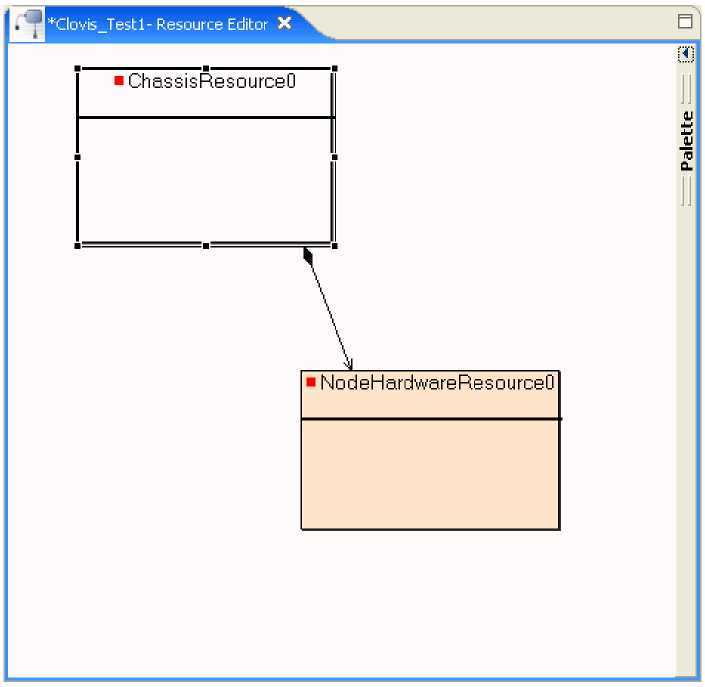


Figure 2-38 Containment Relationship

Note: You can only define valid containment relationships using the composition option. For example, ClovisWorks will not let you define a containment relationship depicting as blades containing a chassis. When you try to do this, the option simply does not work.

2.8 Using Component Editor

The ClovisWorks Component Editor lets you create the components and then define their respective properties. You can create SAF-aware components such as proxied and non-proxied components using the features provided by the editor.

To open the Component Editor, right click the mouse from the ClovisWorkspace and select Component Editor. The Component Editor displays, as shown in Figure 2-39 .

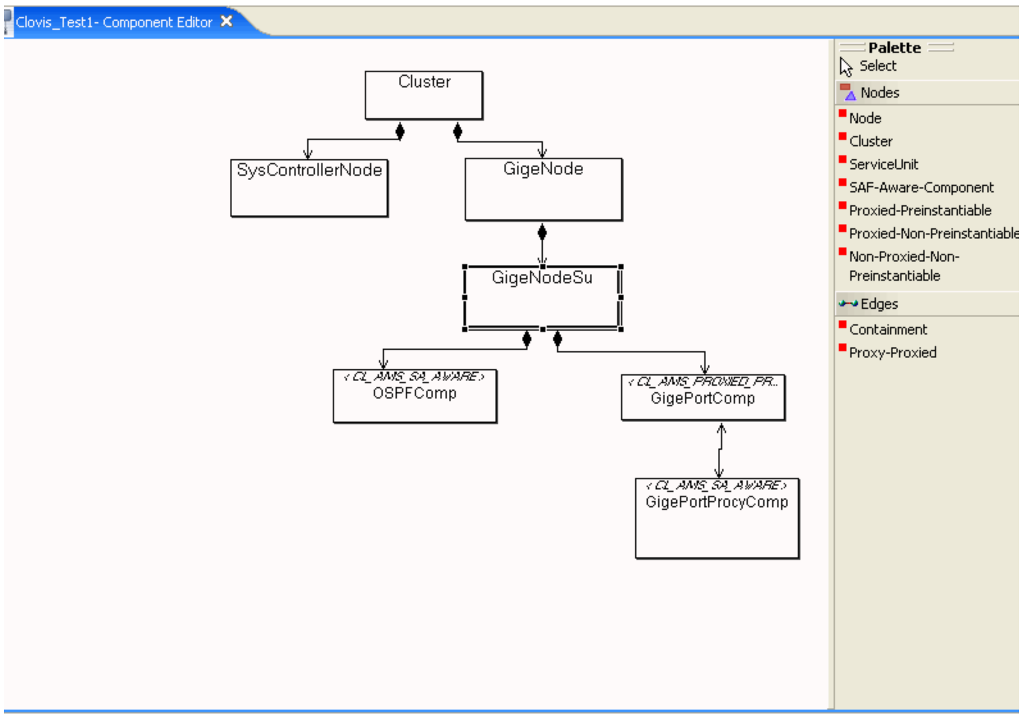


Figure 2-39 Component Editor

In the Component Editor figure, you can see the Component Editor Area and the Palette, which have been appropriately labeled. On the right hand side is the palette that contains the seven default meta classes.

- Node
- Cluster
- Service Unit
- SAF-Aware-Component
- Proxied-Pre-Instantiable
- Proxied-Non-PreInstantiable
- Non-Proxied-Non-PreInstantiable

The ClovisWorks Component Editor offers easy-to-use features that let you create the class type you want and define the relationship between them using the relation connectors. To define the Components for the High Availability, you must drag and drop the required meta class from the Palette Area to the Component Editor Area and then define their attributes and properties. Each pre-defined meta class is readily available for your instant use.

Note: You need to have a corresponding component in the Component Editor for all the resources you have modeled in the Resource Editor. The next section describes in detail each meta class and explains how to use them.

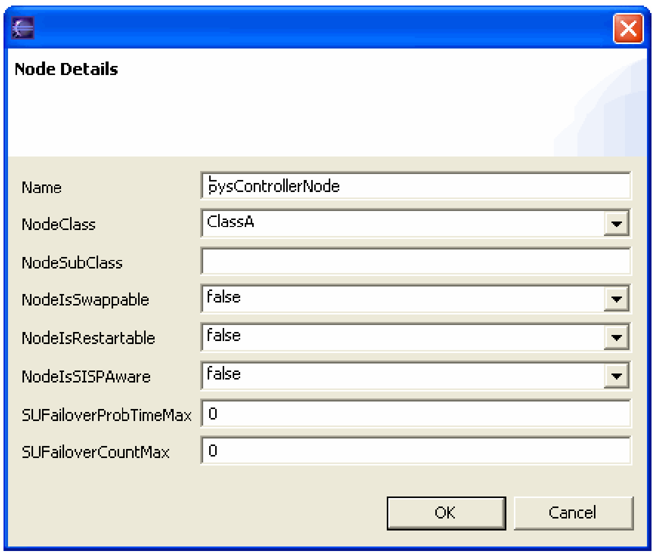
Contents - Index - Back

2.8.1 Node

Node is a logical representation of physical node present in a system. A node can be a computer in a cluster of computers connected together or it can be a blade contained in a chassis. Node meta class captures those attributes of cluster nodes as defined by SAF. After defining the node in the Component Editor, you must name it appropriately and then associate it with a corresponding resource. To represent a Node meta class in the Component Editor, perform the following steps:

1. Click Node in the Palette.
2. Drag the cursor onto the Component Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create the Node.
3. Drag and drop the object in the Component Editor. The object is available in the Editor.

To specify a name for node and other details, double click the node object in the Component Editor. The Node details dialog appears, as shown in Figure 2-40 .


Node Details

Name: sysControllerNode
NodeClass: ClassA
NodeSubClass:
NodeIsSwappable: false
NodeIsRestartable: false
NodeIsSISPAware: false
SUFailoverProbTimeMax: 0
SUFailoverCountMax: 0

Figure 2-40 Node Details

Function	Description
Name	Unique name of the node in the system
NodeClass	This identifies the type of node. There are 4 classes defined A, B, C and D. Refer to Node Class Definitions for more information on each Node class type.
NodeSubClass	Unique identifier for type of node as visible to an administrator. For example, SONET_OC3_V2 or Ethernet_8x100_v3.6.
NodeIsSwappable	Can this node be swapped out or it must be present at all times?

Contents - Index - Back

Function	Description
NodeIsRestartable	If yes, then node is restarted when necessary, if not the entire system is restarted when this node has a fault.
NodeIsSISPAware	Does SISP run on this node
SUFailoverProbTimeMax	The maximum number of times SUs can failover within the probation time, without causing a node failover.
SUFailoverCountTimeMax	If suFailoverCountMax number of service units on a node fail within the suFailoverProbTime interval (SU failover probation time) after the first SU failure, then the fault is escalated to a node failure.

Node Class Definitions

NodeClass A: These cards act as chassis managers or system controllers, and are either standalone or run only SISP controller functions. Customer applications will typically not run on a Class-A card as the intent is to keep them stable and working in a system over a long interval. The Class-A cards are the primary physical link for an external management entity such as EMS. A Class-A blade can assume access to persistent storage on the blade or within the system.

NodeClass B: There are two subtypes within this profile. Class-B1 cards only support SISP master functions and SISP controller functions are located on a separate Class-A card. Class-B2 cards contain both master and controller functions and there is no separate Class-A card in the system. Class B1 blades may or may not have access to local persistent storage and hence must not assume the same while Class-B2 cards may assume access to local persistent storage. Class-B cards may also contain application functions.

NodeClass C: This actually defines a set of card profiles, each of which can contain a different set of SISP functions. However, the common link between all is that they cannot have SISP master or controller functions and are only participants in a SISP system. Class-C blades may or may not have local persistent storage and hence must not assume the same.

NodeClass D: SISP is aware of these blades in the system and must manage them but SISP components do not run on this blade. Note this profile does not have any impact on packaging but is still defined as the profile terminology will also be used within SISP.

To associate a node with a resource, right click the node in the Component Editor and select ‘Associate Resources’. This is illustrated in Figure 2-41 .

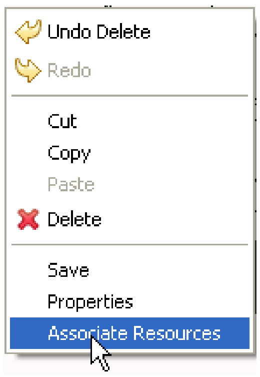


Figure 2-41 Associate Resource Menu

The Associate Resources dialog appears, as shown in Figure 2-42 .

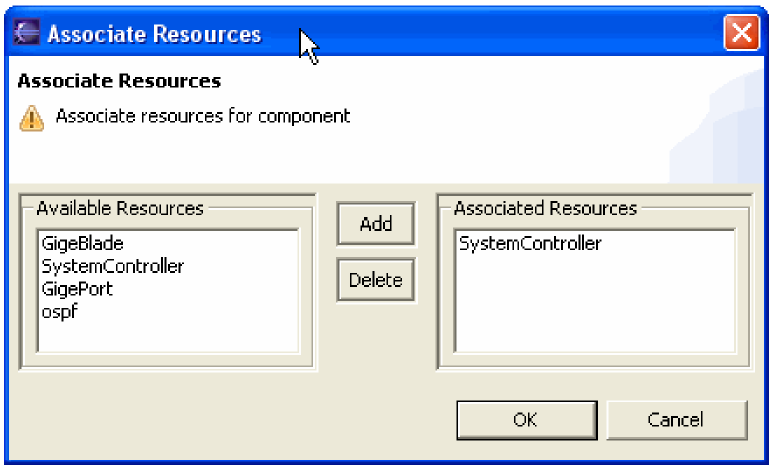


Figure 2-42 Associate Resources

The Associate Resources dialog displays all the available resources. If you have defined the node for a GigeBlade, then select the GigeBlade or the respective resource from the ‘Available Resources’ pane, click ‘Add’ and then click ‘OK’. The resource association is complete.

Note: You cannot associate a cluster or a service unit to a resource from the Component Editor because Cluster is at the very top level like a Chassis while a Service Unit is a logical entity that exists for the definition of redundancies.

2.8.2 Cluster

Cluster is a collection of nodes as defined by Service Availability Forum. The cluster meta class captures those attributes of cluster as defined by SAF MIB. To represent a Cluster meta class in the Component Editor, perform the following steps:

1. Click Cluster in the Palette.
2. Drag the cursor onto the Component Editor Area. Notice the cursor has an additional rectangle and ‘+’ symbol, indicating that you’re ready to create Cluster.

Contents - Index - Back

3. Drag and drop the object in the Component Editor. The object is available in the Editor.

To specify a name for a Cluster, double click and specify a name. You can do the same by selecting the Cluster, right clicking and then selecting properties from the menu.

2.8.3 Service Unit

Service Unit is collection of components which can be switched over to a redundant, identical collection of components. Service unit meta class contains the attributes as defined by SAF MIB for service units. To represent a Service Unit meta class in the Component Editor, perform the following steps:

1. Click Service Unit in the Palette.
2. Drag the cursor onto the Component Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create Service Unit.
3. Drag and drop the object in the Component Editor. The object is available in the Editor.

To specify a name for a Service Unit, double click and specify a name. You can do the same by selecting Service Unit, right clicking and then selecting properties from the menu.

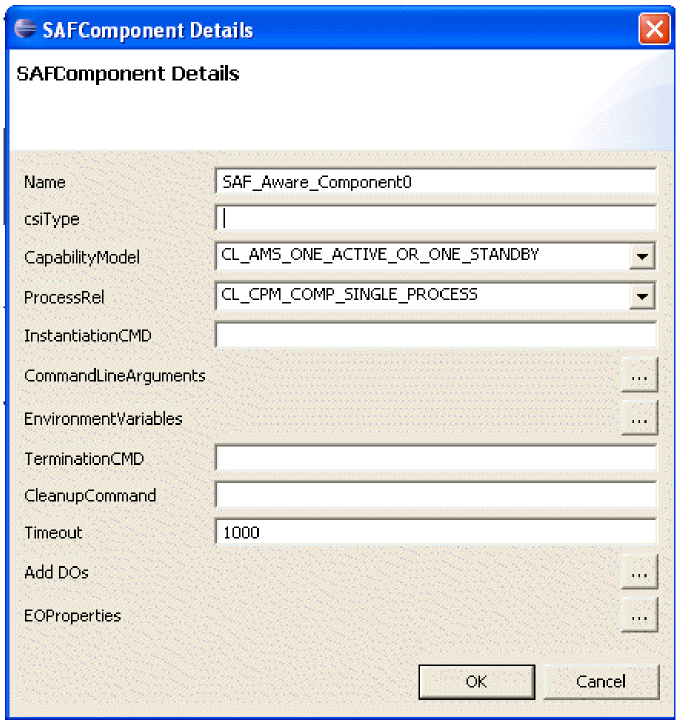
2.8.4 SAF-Aware-Component

An SAF-Aware-Component is one that uses the SAF AMF API to talk to the HA framework. Consequently, the typical example of a SAF aware component is a software process that has been linked with the AMF client library and implements the necessary callbacks. To represent an SAF-Aware-Component meta class in the Component Editor, perform the following steps:

1. Click SAF-Aware-Component in the Palette.
2. Drag the cursor onto the Component Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create SAF-Aware-Component.
3. Drag and drop the object in the Component Editor. The object is available in the Editor.

To specify a name for an SAF-Aware-Component, double click and specify a name. You can do the same by selecting SAF-Aware-Component, right clicking and then selecting properties from the menu. The SAF-Aware-Component details screen appears, as shown in Figure 2-43 .

Contents - Index - Back



Name	SAF_Aware_Component0
csiType	
CapabilityModel	CL_AMS_ONE_ACTIVE_OR_ONE_STANDBY
ProcessRel	CL_CPM_COMP_SINGLE_PROCESS
InstantiationCMD	
CommandLineArguments	...
EnvironmentVariables	...
TerminationCMD	
CleanupCommand	
Timeout	1000
Add DOs	...
EOProperties	...

Figure 2-43 SAF-Aware Component Detail

Name	Name of the SAF component
CSI Type	Denotes the Component service instance type for which this component can provide service.
CapabilityModel	The different component capability defined by AMF
ProcessRel	Whether the Application is of Multi-process, single-process or single threaded. Currently, only single and multi processes are supported.
InstantiationCMD	This value denotes the Instantiate command of the component. This is not applicable for components with saAmfCompProperty value as proxiedPreInstantiable or proxiedNonPreInstantiable.
CommandLineArguments	This option has a separate screen that is documented in the following pages.
EnvironmentVariables	This option has a separate screen that is documented in the following pages.
Termination CMD	This value denotes the Terminate script of the component. This command is applicable for components with saAmfCompProperty value as unproxiedNonSaAware
CleanupCommand	This value denotes the Cleanup command of the component

Timeout	Operations such as instantiation, termination, cleanup must be completed within the specified timeframe for the corresponding components.
Add DOs	This option has a separate screen that is documented in the following pages.
EO Properties	This option has a separate screen that is documented in the following pages.

To specify CommandLineArgument details, click ..., the CommandLineArguments dialog appears, as shown in Figure 2-44 .

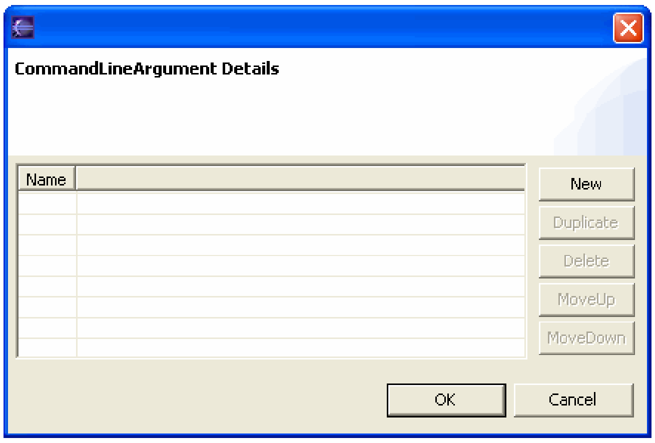


Figure 2-44 CommandLine Argument Details

To specify EnvironmentVariable details, click ..., the Environment Variables dialog appears, as shown in Figure 2-45 .

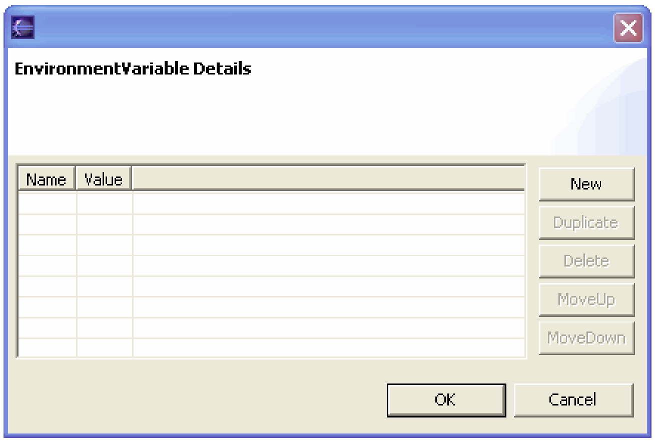


Figure 2-45 Environment Variable Details

To specify the DO properties, click ..., the Device Object details dialog appears, as shown in Figure 2-46 .

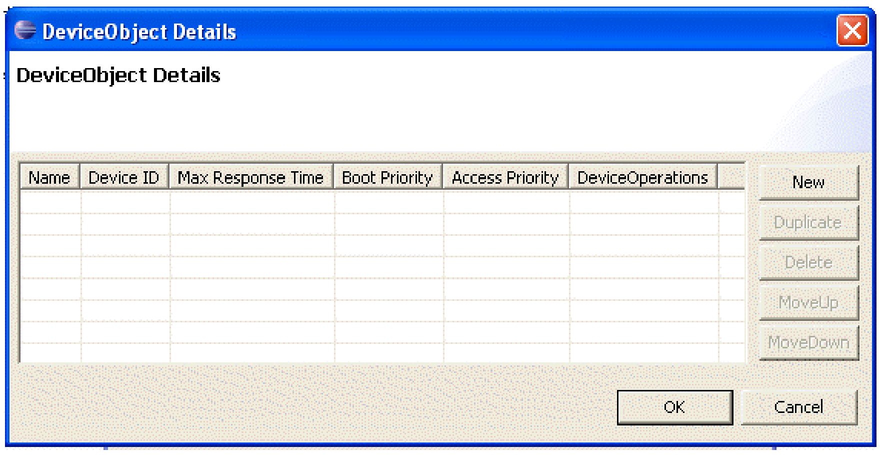
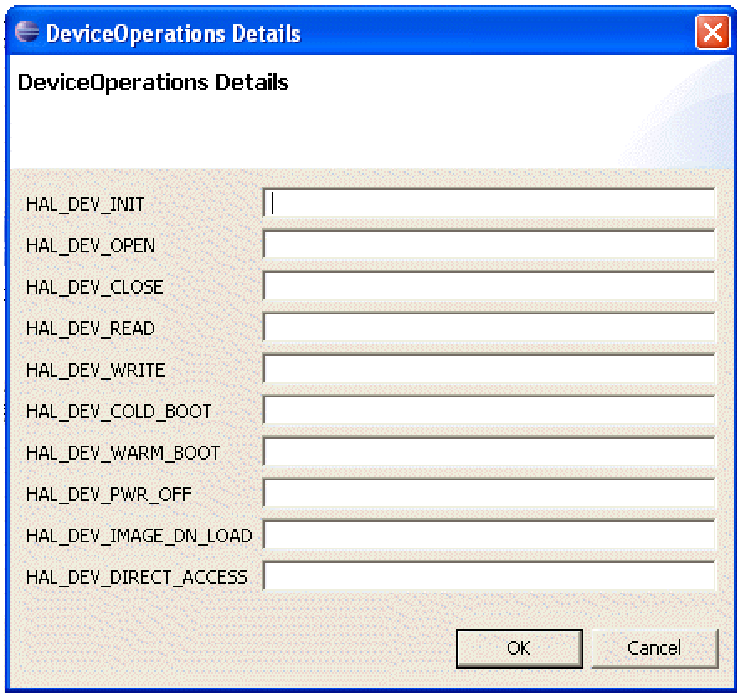


Figure 2-46 Device Object Details

Name	Name of the Device Object
DeviceId	The unique id of the Device Object
MaxResponseTime	Maximum Response Time
BootPriority	Boot priority of the device object
AccessPriority	Access priority of the device object
DeviceOperations	Click ... to go to Device Operations dialog. This option has a separate screen that is documented in the following page.

Contents - Index - Back



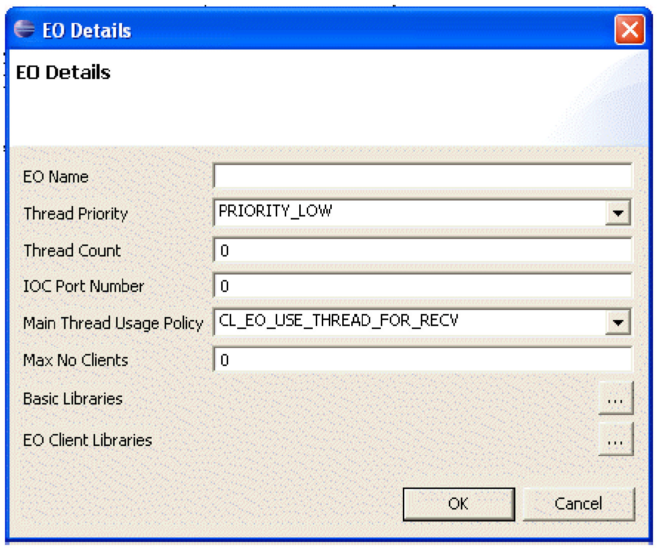
DeviceOperations Details

HAL_DEV_INIT	
HAL_DEV_OPEN	
HAL_DEV_CLOSE	
HAL_DEV_READ	
HAL_DEV_WRITE	
HAL_DEV_COLD_BOOT	
HAL_DEV_WARM_BOOT	
HAL_DEV_PWR_OFF	
HAL_DEV_IMAGE_DN_LOAD	
HAL_DEV_DIRECT_ACCESS	

HAL_DEV_INIT	Function name for device initialization
HAL_DEV_OPEN	Function name for device open
HAL_DEV_CLOSE	Function name for device close
HAL_DEV_READ	Function name for device read
HAL_DEV_WRITE	Function name for device write
HAL_DEV_COLD_BOOT	Function name for device cold boot
HAL_DEV_WARM_BOOT	Function name for device warm boot
HAL_DEV_PWR_OFF	Function name for device power off
HAL_DEV_IMAGE_DN_LOAD	Function name for device image download
HAL_DEV_DIRECT_ACCESS	Function name for device' direct access

To specify the EO properties, click ..., the EO Details dialog appears, as shown in figure Figure 2-47 .

Contents - Index - Back

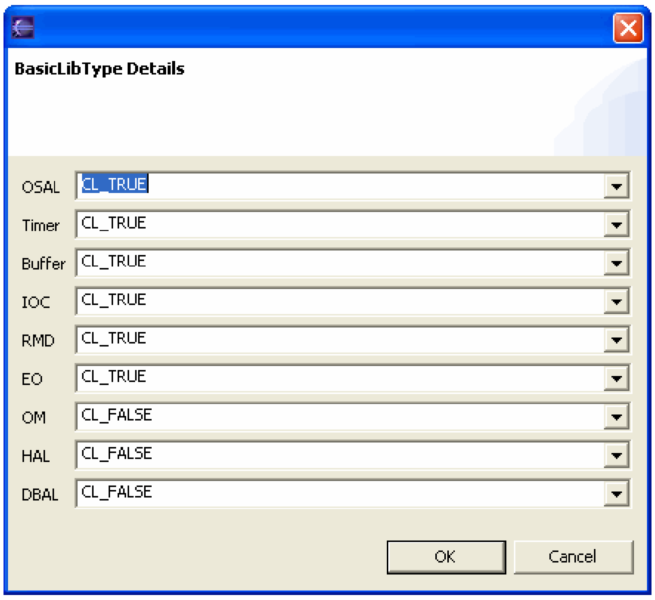


EO Name:
Thread Priority: PRIORITY_LOW
Thread Count: 0
IOC Port Number: 0
Main Thread Usage Policy: CL_EO_USE_THREAD_FOR_RECV
Max No Clients: 0
Basic Libraries: ...
EO Client Libraries: ...

Figure 2-47 EO Details

EO Name	Name of the execution object
Thread Priority	Indicates the EO thread priority.
Thread Count	Indicates the number of RMD threads.
IOC Port Number	This is the requested IOC communication port.
Main Thread Usage Policy	Policy for the main thread usage.
Basic Libraries	The first six libraries must always be set to TRUE. Rest of them can be set based on the requirement of the application. When you set them to TRUE, the corresponding libraries will be automatically initialized. (See Figure 2-48 .)
EO Client Libraries	These libraries are optional. If they are set to TRUE, the corresponding libraries will be automatically initialized. (See Figure 2-49 .)

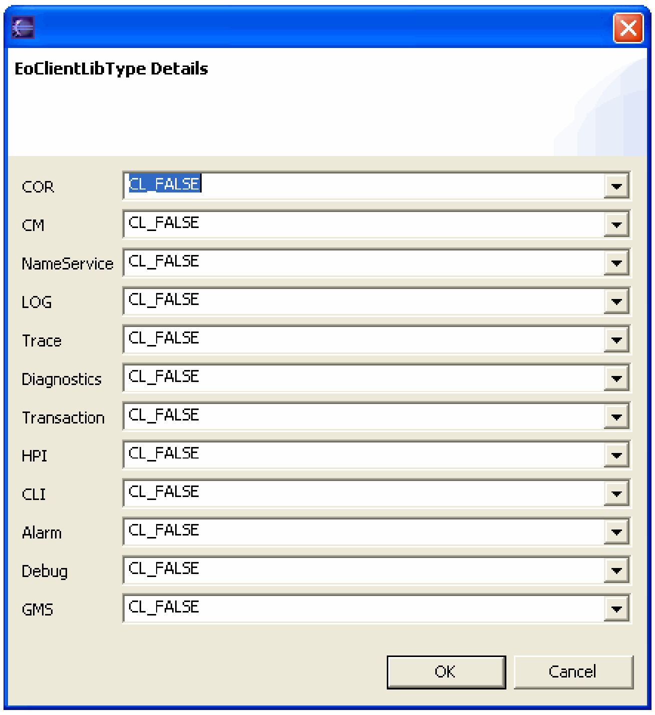
To specify the details of Basic Lib, click ..., the BasicLibType Details dialog appears, as shown in figure Figure 2-48 .


BasicLibType Details

OSAL	CL_TRUE
Timer	CL_TRUE
Buffer	CL_TRUE
IOC	CL_TRUE
RMD	CL_TRUE
EO	CL_TRUE
OM	CL_FALSE
HAL	CL_FALSE
DBAL	CL_FALSE

Figure 2-48 Basic Lib Type Details

To specify the details of Client Lib, click [...], the EoClientLibType Details dialog appears, as shown in figure Figure 2-49 .


EoClientLibType Details

COR	CL_FALSE
CM	CL_FALSE
NameService	CL_FALSE
LOG	CL_FALSE
Trace	CL_FALSE
Diagnostics	CL_FALSE
Transaction	CL_FALSE
HPI	CL_FALSE
CLI	CL_FALSE
Alarm	CL_FALSE
Debug	CL_FALSE
GMS	CL_FALSE

Figure 2-49 Client Lib Type Details

Contents - Index - Back

2.8.5 Proxied Preinstantiable Component

A proxied component is one that does not talk directly to the SAF AMF API but is still managed by the HA framework. This management takes place through a proxy that does talk directly to the HA infrastructure, thus the proxy is a SAF-aware-component. A preinstantiable component is one that can be started and kept in an idling mode without assigning any workload to it.

A proxied-preinstantiable (p-p) is a combination of the above— it does not talk directly to the HA infrastructure but whose work assignment can be controlled by the HA infrastructure via the proxy. An example of this would be a software component that can be kept idle after starting up, but which has not been linked with the AMF client lib and so must be controlled via a proxy. To represent a Proxied-Preinstantiable Component meta class in the Component Editor, perform the following steps:

1. Click Proxied-Preinstantiable in the Palette.
2. Drag the cursor onto the Component Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create Proxied-Preinstantiable.
3. Drag and drop the object in the Component Editor. The object is available in the Editor.

To specify a name for Proxied-Preinstantiable, double click and specify a name. You can do the same by selecting Proxied-Preinstantiable, right clicking and then selecting properties from the menu. The Proxied-Preinstantiable component details screen appears, as shown in Figure 2-43 .

Note: The procedure to specify the component details is the same as that of SAF-Aware-Component details except that you don't need to specify the EO details for a Proxied-Preinstantiable component. Hence, the EO details option will not be available to you in the Component details screen. For specifying information related to component details, refer to SAF-Aware-Component section.

2.8.6 Proxied Non-Preinstantiable Component

A Proxied-Non-Preinstantiable (p-np) component is the same as proxied-preinstantiable, except that it cannot be kept in an idle state. An example would be a hardware component that starts providing when it comes on and the only way to control it is via a proxy (that is SAF-aware). The common thing about both (Proxied-Preinstantiable and Proxied-Non-Preinstantiable) is that because of the SAF aware proxy it is possible to control the components in a more fine grained way. To represent a Proxied-Non-Preinstantiable Component meta class in the Component Editor, perform the following steps:

1. Click Proxied-Non-Preinstantiable in the Palette.
2. Drag the cursor onto the Component Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create Proxied-Non-Preinstantiable.
3. Drag and drop the object in the Component Editor. The object is available in the Editor.

Contents - Index - Back

To specify a name for Proxied-Non-Preinstantiable, double click and specify a name. You can do the same by selecting Proxied-Non-Preinstantiable, right clicking and then selecting properties from the menu. The Proxied-Non-Preinstantiable component details screen appears, as shown in Figure 2-43 .

The procedure to specify the component details is the same as that of SAF-Aware-Component details except that you don't need to specify the EO details for a Proxied-Non-Preinstantiable component. Hence, the EO details option will not be available to you in the Component details screen. For specifying information related to component details, refer to SAF-Aware-Component section.

2.8.7 Non-Proxied Non-Preinstantiable Component

This is a component that is not SAF-aware which does not have a proxy but can be directly controlled by the HA framework. (np-np). For example, this is like a fan tray that once you add to the system, it starts working, but cannot be kept in an idle state. The HA infrastructure can control it in a very coarse way, start it, stop, restart and so on, but nothing more than that. To represent a Non-Proxied-Non-Preinstantiable Component meta class in the Component Editor, perform the following steps:

1. Click Non-Proxied-Non-Preinstantiable in the Palette.
2. Drag the cursor onto the Component Editor Area. Notice the cursor has an additional rectangle and '+' symbol, indicating that you're ready to create Non-Proxied-Non-Preinstantiable.
3. Drag and drop the object in the Component Editor. The object is available in the Editor.

To specify a name for Non-Proxied-Non-Preinstantiable, double click and specify a name. You can do the same by selecting Non-Proxied-Non-Preinstantiable, right clicking and then selecting properties from the menu. The Non-Proxied-Non-Preinstantiable component details screen appears, as shown in Figure 2-43 .

The procedure to specify the component details is the same as that of SAF-Aware-Component details except that you don't need to specify the EO details for a Non-Proxied-Non-Preinstantiable component. Hence, the EO details option will not be available to you in the Component details screen. For specifying information related to component details, refer to SAF-Aware-Component section.

2.8.8 Containment

After creating the Components, you must define the containment relationship between the components. Containment enables a component to contain another component and access the contents of the contained object thus reducing the overhead of redefining the properties for the object independently.

Perform the following steps to define containment relationships:

- Click **Containment** in the Palette. The cursor changes to an arrow with an additional circle.

Contents - Index - Back

- Click the Component, hold down the left mouse button and drag the mouse pointer to the other Component, the content of which you wish to contain, and click the mouse pointer again and leave. A containment arrow is added creating the containment relationship between the two components.

2.8.9 Proxy-Proxied

Proxy-Proxied relationship is an association relationship between proxy and proxied component. Using this relationship, a proxy component can know about what component it is proxy for and vice versa.

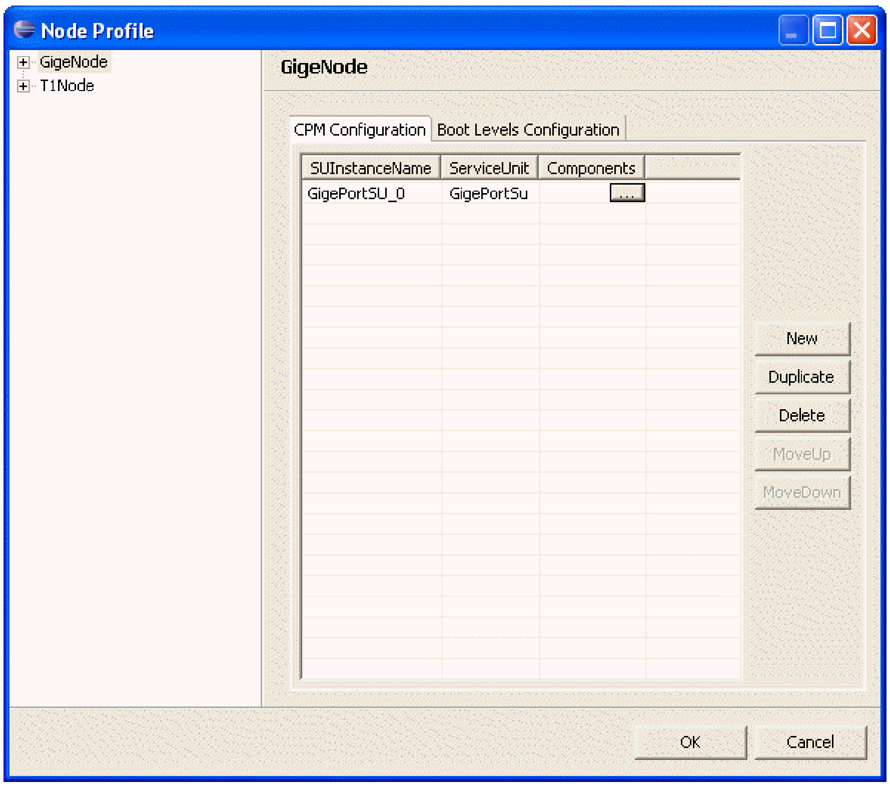
Perform the following steps to define Proxy-Proxied relationship:

- Click **Proxy-Proxied** in the Palette. The cursor changes to an arrow with an additional circle.
- Click the Component, hold down the left mouse button and drag the mouse pointer to the other Component to define the Proxy-Proxied relationship between two components.

Contents - Index - Back

2.9 Using Node Profile

You can specify Node profiles using the option from the Clovis submenu. To launch the Node Profiles dialog, right click in the Clovis Workspace, select Clovis and then select Node Profiles. The Node Profiles dialog appears, as shown in Figure 2-50 .



SUInstanceName	ServiceUnit	Components
GigePortSU_0	GigePortSu	...

Figure 2-50 CPM Configuration Details

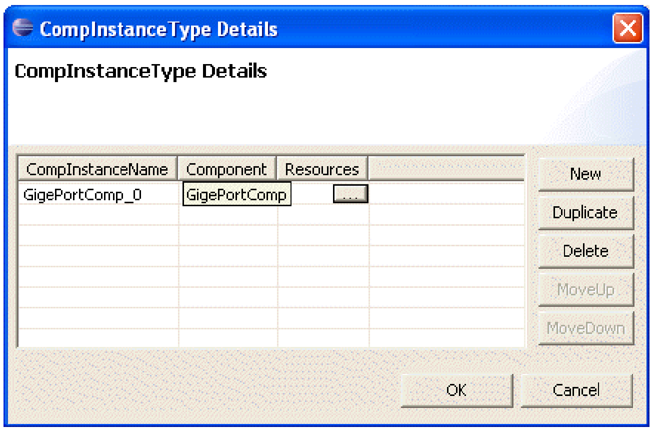
2.9.1 Specifying CPM Configuration

To specify CPM configuration details, click CPM configuration in Figure 2-50 . The CPM configuration screen appears. Specify the CPM configuration for the selected node.

Instance Name	Name of the service unit instance
Service Unit	Name of the type (class) of the service unit.
Components	Component Instance Type details

Specifying Component Instance Type Details

Click ... to launch the Component Instance Type details dialog, as shown in Figure 2-51 .



CompInstanceName	Component	Resources
GigePortComp_0	GigePortComp	...

Figure 2-51 Component Instance Details

Component Instance Name	Name of the component instance.
Component	Name of the Component Type.
Resource	Click ... to specify the Resource Instance details.

The Resource Type Details dialog is shown in Figure 2-52 .

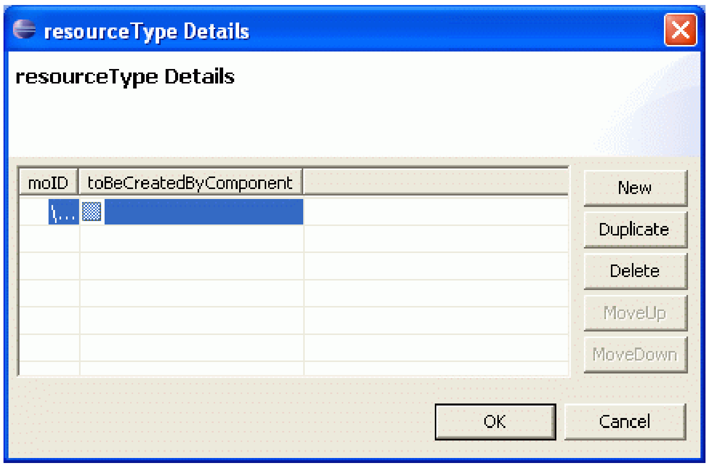
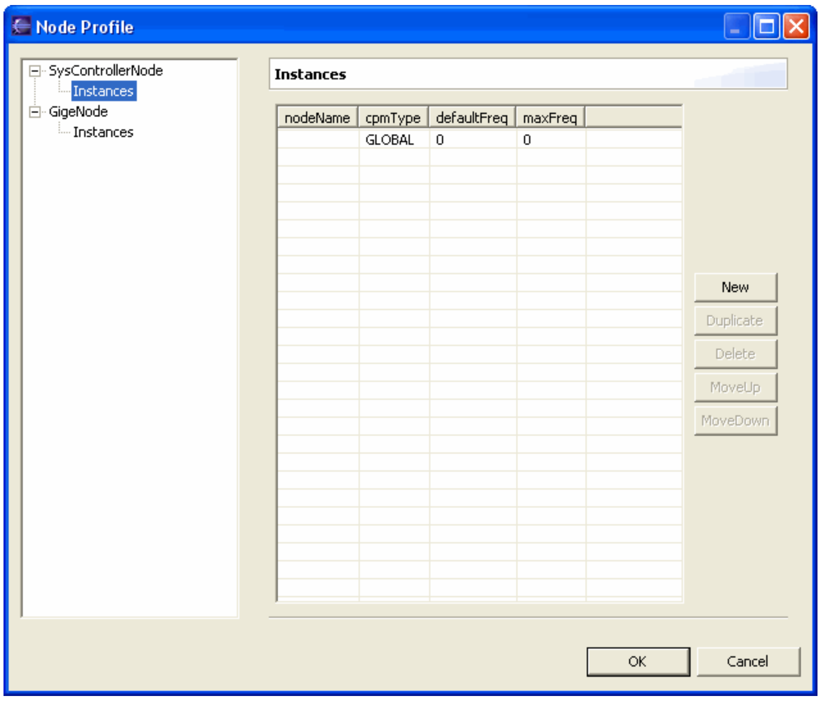


Figure 2-52 Resource Type Details

Contents - Index - Back

MoId	Unique identifier of the managed object
Tobecreated by component	If this is selected, the instance will be created by the component in COR.

After specifying the component instance details, go to the Instances dialog and specify the number of instances to this respective System Controller Node, as shown in Figure 2-53 .



nodeName	cpmType	defaultFreq	maxFreq
	GLOBAL	0	0

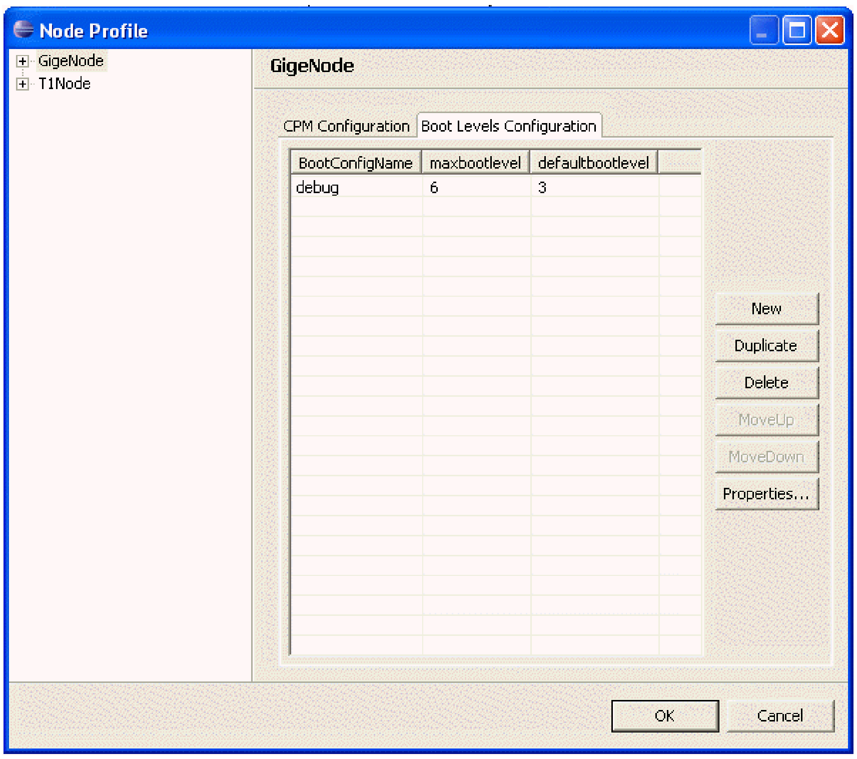
Figure 2-53 Node Instance Details

Node Name	Name of the Node
CPM Type	Global or Local CPM. Global CPM available on a master card and CPM local will be available on all the cards.
default frequency	Frequency by which CPM will poll all the EOs in the system for healthcheck
max frequency	Maximum frequency by which CPM will poll the EOs in the system for healthcheck.

Contents - Index - Back

2.9.2 Specifying Boot-level Configuration

To specify Boot-level configuration details, click the Boot-levels configuration in Figure 2-54 . The Boot-level configuration screen appears. Specify the Boot-level configuration for the selected node.

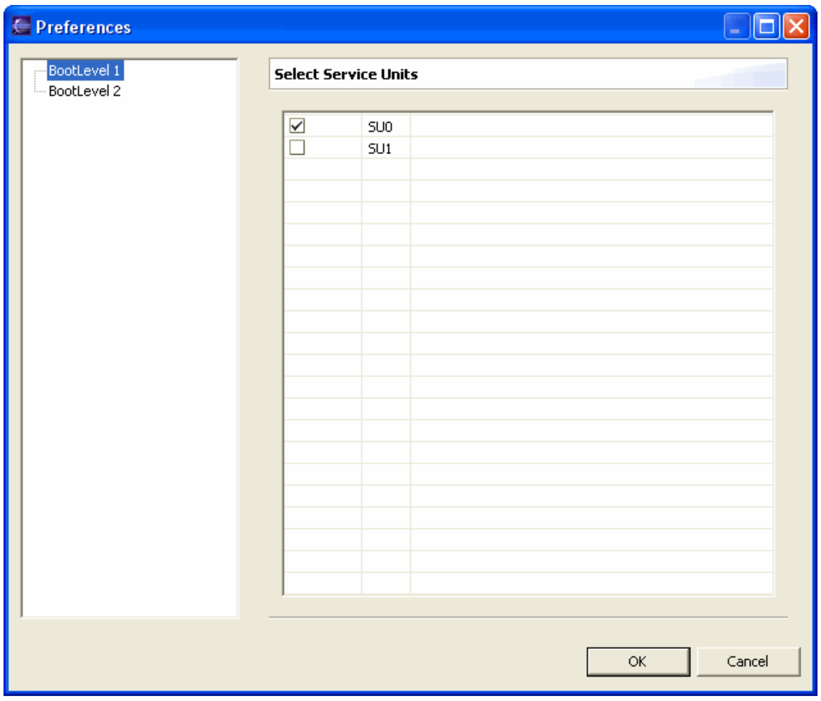


BootConfigName	maxbootlevel	defaultbootlevel
debug	6	3

Figure 2-54 Boot-level Configuration Details

Boot Config Name	The name of the boot profile
maxbootlevel	Maximum number of bootlevel available in the corresponding boot profile
defaultbootlevel	CPM tries to start till the default boot level when it boots up.

Once you have defined the boot configuration details, you must specify the properties. To specify the properties, click Properties. The Preferences dialog appears, as shown in



✓	SU0
☐	SU1

Figure 2-55 Boot Preferences

In the Boot Preferences dialog, select the respective service unit and click OK. If there are any instances, go to the Instances dialog and specify the instances as described in the Specifying CPM Configuration.

2.10 Update Script and Build Files

This option from the Clovis' submenu enables you to sync up to the latest scripts available with the product.

2.11 The Outline View

Outline View represents the hierarchical relationships among multiple Resources. The containment hierarchy can be captured in the Outline View and the relationships can be viewed in a hierarchical structure.

To launch the Outline View, go to Window ->Show View -> Other. From the ShowView dialog, expand the Basic folder. Select Outline and click 'OK'. The Outline View displays, as shown in Figure 2-56 .

Contents - Index - Back

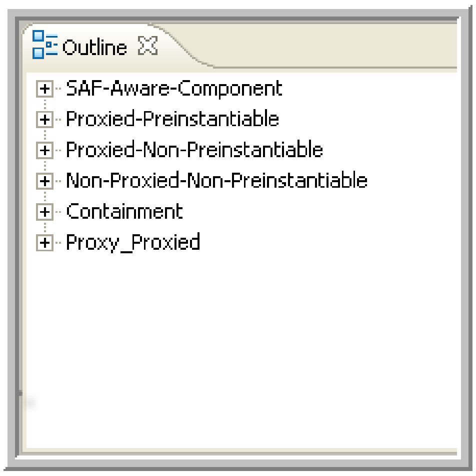


Figure 2-56 Outline View

2.12 The MIBTree View

To launch the MIBTree View, go to Window ->Show View -> Other. From the ShowView dialog, expand the Clovis folder. Select MIB File Explorer and click ‘OK’. The MIB File Explorer displays, as shown in Figure 2-57 .

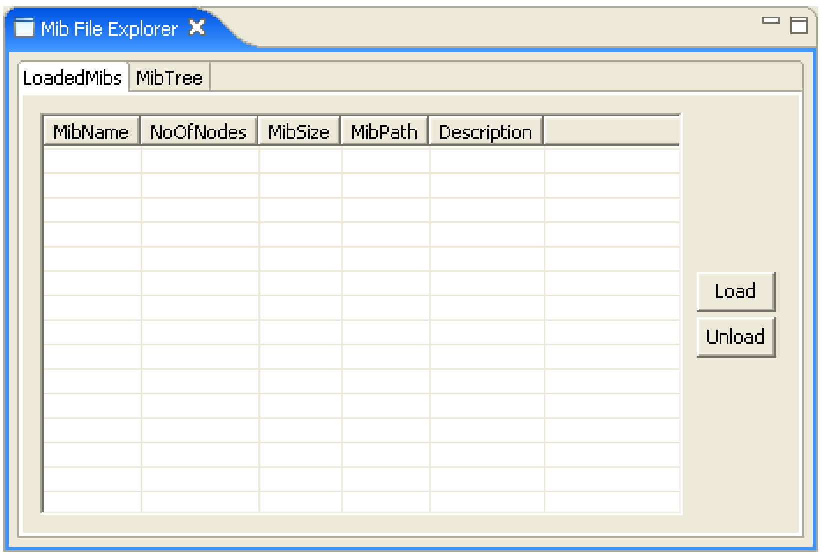


Figure 2-57 MIB File Explorer

The MIB File Explorer has two tabs. For more information on using the two tabs in MIB File Explorer, refer to Importing from a MIB.

Chapter 3
Information Modeling

This Chapter covers the following topics:

3.1 Introduction to Information Modeling

An Information Model (IM) presents a unified view of different components of a system and hides hardware and software running on a system from an end user. An IM facilitates exchange of management information in a platform-independent and technology-neutral way. IM is an object-oriented model that captures essential information about a network element such as its hardware, software and services. Besides, it also captures vital data contained in a system and the relationship between different components. The data could be some configurable parameters of hardware present in the system or configurable parameters of software running on different CPUs in a system. Some of the other attributes the IM contains are:

- State of different hardware/software components.
- Behavior of different software components.
- Alarms and Faults present in a system.
- Performance of different entities in a system.
- Resource utilization of hardware and/or software resources present in a system

Conceptually, IM is somewhat similar to DMTF's Common Information Model (CIM). IM's object-oriented approach makes it easier to track the relationships and inter dependencies between managed objects. The DMTF CIM is an approach to the management of systems and networks that applies to the basic structuring and conceptualization techniques of the object-oriented paradigm. The approach uses a uniform modeling formalism that together with the basic repertoire of object-oriented approach, constructs and supports the cooperative development of an object-oriented schema across multiple organizations.

IM also provides information about instances of classes and their relationships. It provides a common and standard protocol to describe an object to all the Clovis components and provides a scheme to name each object, its attributes, its relationships, and convert it into an IOC address.

An Information Model includes:

1. Network Elements, such as:
   - *Hardware*:
     Chassis, blade, system controller, interface port, logical port, logical

Contents - Index - Back

connections such as ATM VPC, VLAN, and so on.

- *Software:*

  Application protocols such as OSPF, SS7, and so on.

A Clovis Information Model (IM) is a generic framework or an abstract representation of the entities in a managed environment. It abstracts the properties, operations, and relationships of the Resources and Components in the managed environment. In order to manage any hardware resource or a software resource or anything that you require to manage, you must represent it as a ***Resource*** in the Clovis IM. The Information Modeling capability of ClovisWorks provides a mechanism to represent all software and hardware resources as an overall system. The resources represented in the Information Model will contain the actual and desired states of the physical resource.

In other words, we can also define the Clovis IM as a mechanism that describes the characteristics of the network element for which the infrastructure software is being implemented. IM provides a unified view and association of all the physical and logical objects in the managed environment. Information Modeling transports the current content or state of a Clovis Object such as a blade, port, or any other logical object between different Execution Objects.

Information Modeling is the first step towards the generation of customized *System Infrastructure Software Platform* using **ClovisWorks**. An Information Model is the blueprint of both the software and hardware Resources in a system. ClovisWorks helps build your information model with the help of two intuitive editors— the **Resource Editor** and **Component Editor**. It provides a graphical editing interface that enables you to capture the model information through UML notations. These designs are saved as XML files. The following section in this chapter describes an end-to-end flow (from creating an information model to generating SISP-compatible code and deploying it onto the SISP).

3.2 End-to-End Flow Using ClovisWorks

This section describes the end-to-end flow, right from creating the information model to generating code and then linking it to the SISP libraries. The Figure 3-1 illustrates the flow at a high level.

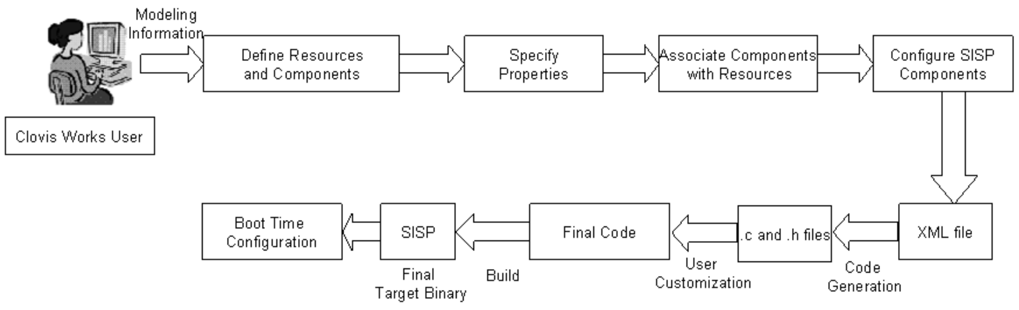


Figure 3-1 ClovisWorks Flow

The System Model (The NE to be modeled) forms the input for ClovisWorks. As a user of ClovisWorks, you will define an Information Model of the system using the intuitive GUI and various other features of the tool. The first step in this process is to define the various hardware and software objects of the system to be modeled as Resources in ClovisWorks. And then specify the properties of both software and hardware entities that have been modeled. During this stage, you can also import an SNMP MIB (management information base) to define the attributes of a standard managed object.

The next step in the process is defining the SAF (Service Availability Forum) model by creating corresponding components for each resource and then associating them with the respective resources. You then need to configure Build time SISP components and save the data. The data will automatically be saved in XML files. These data files form the input for code generation. Using ClovisWorks, you can easily generate code for the modeled Network Element.

ClovisWorks generates platform-specific code depending upon the specified build time configuration. Based on your requirement, you can modify the source files to add additional functionality to the platform-specific code as well as modify code to suit your Policy requirements.

ClovisWorks also provides the capability to specify boot time configurations from the GUI itself. The system deployer can specify boot time configuration based on the need and generate an XML data file. This XML data file will contain common information, which could be edited for further modification. For example, let us assume that a Chassis is present in your model that has 4 line cards in it with different IP addresses. You can edit the boot time configuration XML file to specify the transport addresses of these cards, which could be something very specific to your application. Following the modification, you can integrate the code with SISP libraries and deploy the executable onto the hardware.

ClovisWorks graphical development environment complements SISP by greatly simplifying the system definition, development, debugging and deployment of SISP-based software. When you create an Information Model you might be required to configure SISP components based on the type of Resources you might have modeled.

3.3 End-to-End Flow without Using ClovisWorks

ClovisWorks simplifies the process of generating code that SISP can consume. The previous immediate section clearly explained the entire process. In the event of users purchasing SISP without ClovisWorks, they will not get the advantage of rapidly generating code that SISP can consume and yet they will be required to manually generate code. The following diagram illustrates the process of manually writing the code:

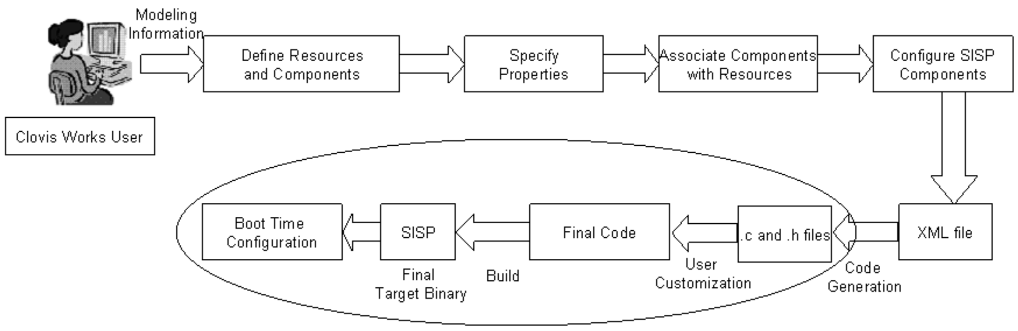


Figure 3-2 Flow without using ClovisWorks

The flagship product from Clovis Solutions is SISP, which can be purchased with or without the development framework ClovisWorks.

The Figure 3-2 precisely illustrates the process that a user must follow to manually produce the SISP-compliant code. Every SISP component, such as Boot Manager, Chassis Manager, Fault Manager and so on will have configuration files such as .h and .c files. Users must manually write code and configure the SISP components using a host of Debug CLI commands. Refer to SISP User Guide for more information on how to use Debug CLI commands.

The following diagram provides you a snapshot view of the entire process. The end result is generation of A and B in the following diagram, which is required to be ported onto the SISP platform for the software customized for the targeted application.

Contents - Index - Back

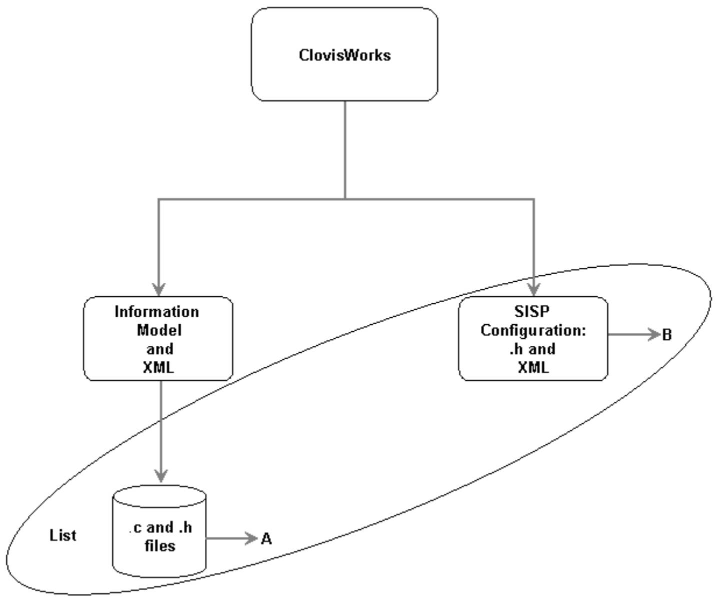


Figure 3-3 SISP Consumption Items

Whether you generate it using ClovisWorks or manually, all you require is A and B eventually to produce software customized for the target application, as shown in Figure 3-3 .

3.4 Sample Network Element

This chapter will guide you through the steps to create an information model for a sample NE. The objective of the sample application is to provide continues streaming of video data to the customers who want to see the movie without an interruption. For this you need to model one System Controller Card and two Blades having a few ports.

You will use ClovisWorks to model the sample network element discussed above. You will represent one model for each resource. You will be modeling the following resources. The primary objective of modeling is to manage these resources:

- Chassis
- System Controller
- GigE Blade
- T1 Blade
- Port

In this tutorial you will create a project named SampleModel-1 and perform various steps to model the resources contained within the sample model to generate files. At a high level, you must perform the following steps to model the resources and generate the files.

1. Create a Project
2. Use Resource Editor

Contents - Index - Back

3. Use Component Editor
4. Specify Build-time Configuration
5. Specify Boot-time Configuration
6. Build Project

3.4.1 Creating Project

The first step that you will be required to model these resources is creating a project in ClovisWorks and then define some important settings.

Perform the following steps to launch ClovisWorks and then define the settings:

1. Launch ClovisWorks
2. Go to the File Menu, select New Project
3. From the resulting dialog, select Clovis System Project and click Next.
4. In the Project dialog, specify the name as SampleModel-1and click Finish. The project is now visible in the Clovis Workspace.

If you want to store your project at a location of your choice, uncheck the Use Default checkbox and specify a location. Click Finish.

Important!: One of the very important steps that you must prudently perform is specifying the SISP location during the start of the project. This is the location where ClovisWorks generates the files for SISP-consumption.

Specifying SISP File Location

Perform the following steps to specify the SISP file location:

1. Right-click from the Clovis Workspace and select Properties. The Clovis System Project dialog appears, as shown in Figure 3-4 .

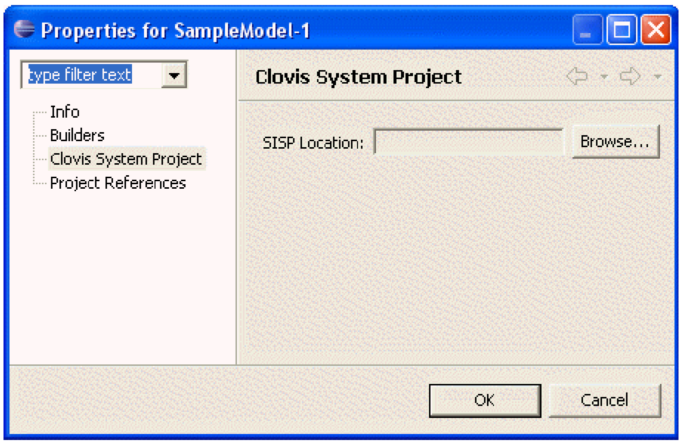


Figure 3-4 SISP Location Dialog

2. Select Clovis System Project to view SISP Location field.

Contents - Index - Back

3. Click Browse and specify the SISP location.

3.4.2 Using Resource Editor

You will use Resource Editor to model the resources and define their relationships. Go to the Resource Editor and use the ready-made meta classes to model the resources. For more information on defining the resources in the Resource Editor, refer to Using Resource Editor.

After you model the resources and define their containment hierarchy, the sample model will appear as shown in Figure 3-5 .

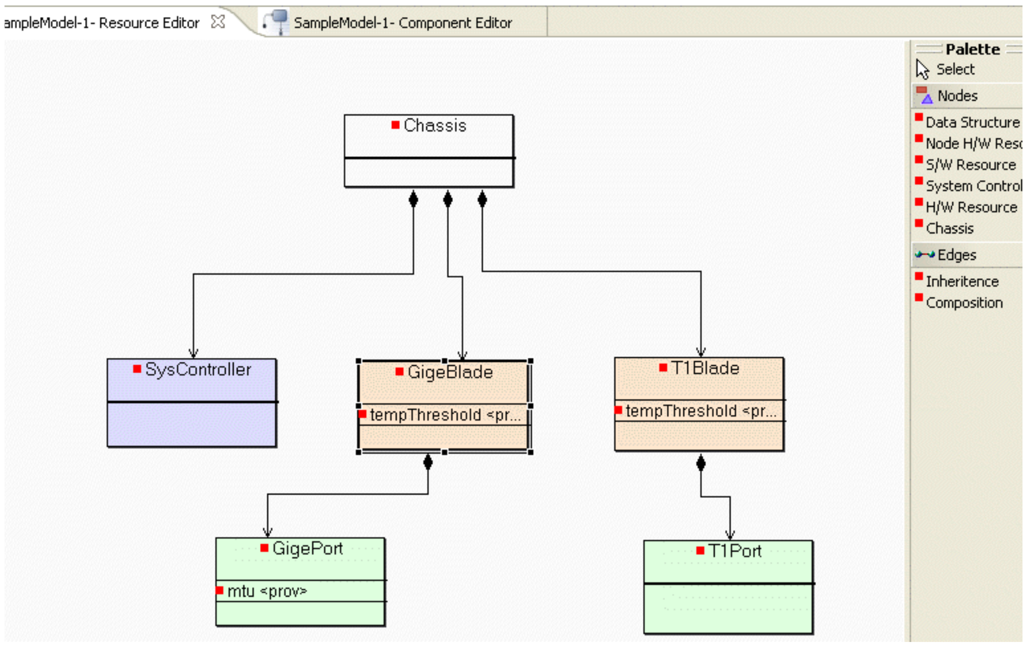


Figure 3-5 Sample Model in Resource Editor

Since a Chassis always contains System Controllers and Blades, you have defined the containment hierarchy as Chassis containing the System Controller and the two blades— GigE and T1. Since these blades contain the ports, you have accordingly defined the containment hierarchy.

The next step in the process is defining their attributes. You define attributes such as Speed or Maximum Transmission Unit (MTU) basically for provisioning. To define the attributes, refer to Using Resource Editor. After defining the attributes, you need to associate Alarms based on your need. If you require Alarms, you must enable them. To enable the Alarms, refer to Node Hardware Resource section.

Note: The Alarms must be predefined for them to be available in the Properties dialog. To define the Alarms, refer to the section Defining Alarm Profiles.

The next step in the process is using the Component Editor.

Contents - Index - Back

3.4.3 Using Component Editor

The primary objective of using the Component Editor is to define the software entities. You define the software entities because they result in the executable. In the Resource Editor, you modeled a Chassis, one System Controller, two Blades and ports. For each of this you need to have a respective component in the Component Editor that realizes the resource modeled in the Resource Editor.

For Chassis you have Cluster, Nodes for Blades and an SA-aware component for the ports. When you model the respective components in the Component Editor, the SampleModel-1 will appear as shown in Figure 3-6 .

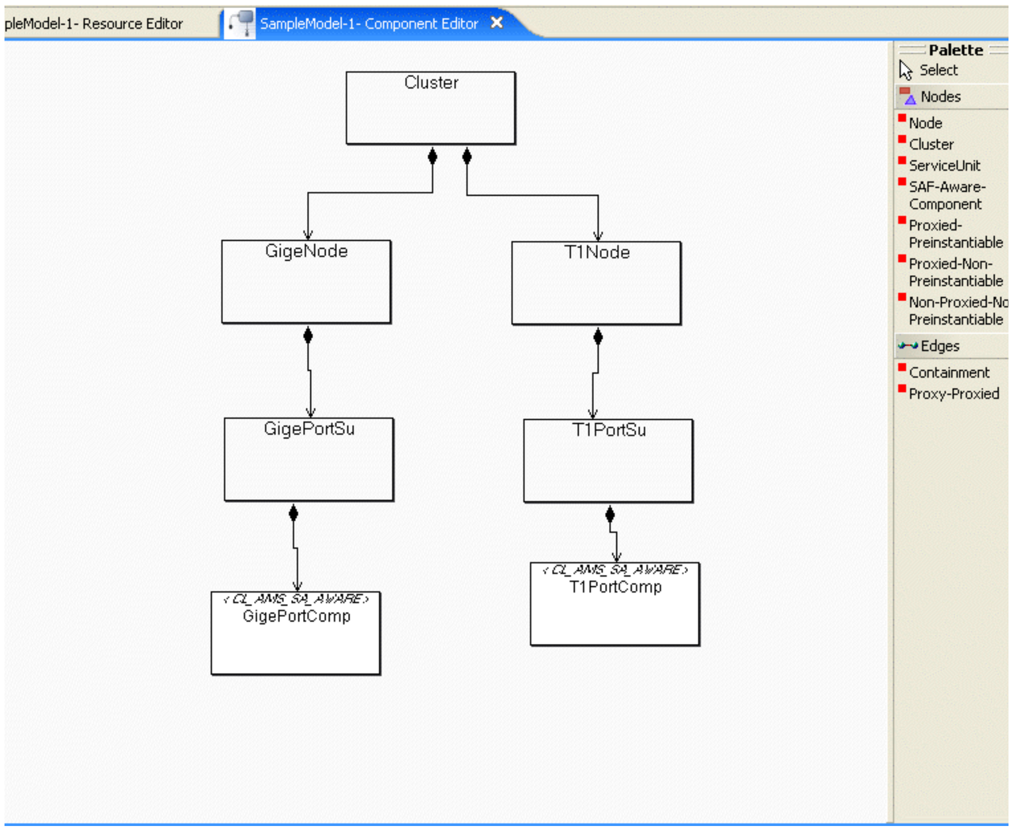


Figure 3-6 Sample Model in Component Editor

Important!: In addition to the respective components that realize a resource, you also see Service Units in the above diagram. A Service Unit is a logical entity that will be created for redundancies. This is as per the SAForum recommendations.

To define the attributes and properties of the components and more information on each type, refer to Using Component Editor section.

Contents - Index - Back

3.4.4 Specifying Build-time Configuration

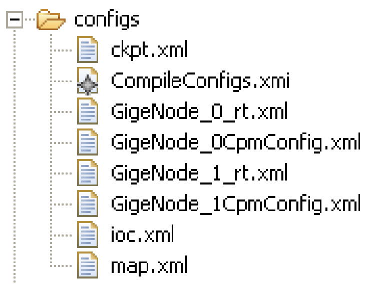
The next step in this process is specifying the build time configuration. You must define the configuration parameters of SISP components. These parameters can not be changed because the parameters you specify here will get embedded into the code.

To specify the Build-time configuration, refer to Configuring Build-time Parameters section. After specifying the parameters, click OK. The respective configuration files are generated in the Config folder.

3.4.5 Specifying Boot-time Configuration

Now, you must define the Boot-time parameters. The parameters that you specify here are configured during boot time based on which ClovisWorks generates files.

To specify the Build-time configuration, refer to Configuring Boot-time Parameters section. After specifying the parameters, click OK. The respective configuration files are generated in the Config folder.



```
configs
    ckpt.xml
    CompileConfigs.xmi
    GigeNode_0_rt.xml
    GigeNode_0CpmConfig.xml
    GigeNode_1_rt.xml
    GigeNode_1CpmConfig.xml
    ioc.xml
    map.xml
```

Figure 3-7 CW Configuration Files

The following is a list of generated files in the

3.4.6 Building Project

The last step in the process of modeling is building the project. To build the project, right-click in the Clovis Workspace and select ‘Build Project’.

Once you build the project, the ClovisWorks-generated code will be available in the SISP location that you specified during the start of this project. The following folders are automatically generated:

- APP
- Config
- SNMP
- Target

Only APP and Config contain the generated files.

Contents - Index - Back

Note: If the Build Project option is not available, go to the Project menu, deselect the option ‘Build Automatically’ and perform the above step again.

Chapter 4
Configuring Components

This chapter covers the following topics:

4.1 Introduction to Component Configuration

The Clovis SISP is a generic engine that can be customized and configured easily to be executed on your platform and support your application software. SISP provides various data structures, enums, and hash defined values to configure the different SISP components. You can configure SISP components during Boot Time and Build Time.

4.1.1 Opening Component Configuration List in ClovisWorks

The next step after creating the Resources and Components and defining their properties for a NE is to configure the SISP components in the ClovisWorks GUI.You can configure the SISP components during boot time and build time.

Configuring Boot-time Parameters

To configure Boot time parameters, do the following:

Right-click from the Clovis Workspace and select 'Boot Configurations'. The 'Boot Configurations' dialog appears, as shown in Figure 4-1 .

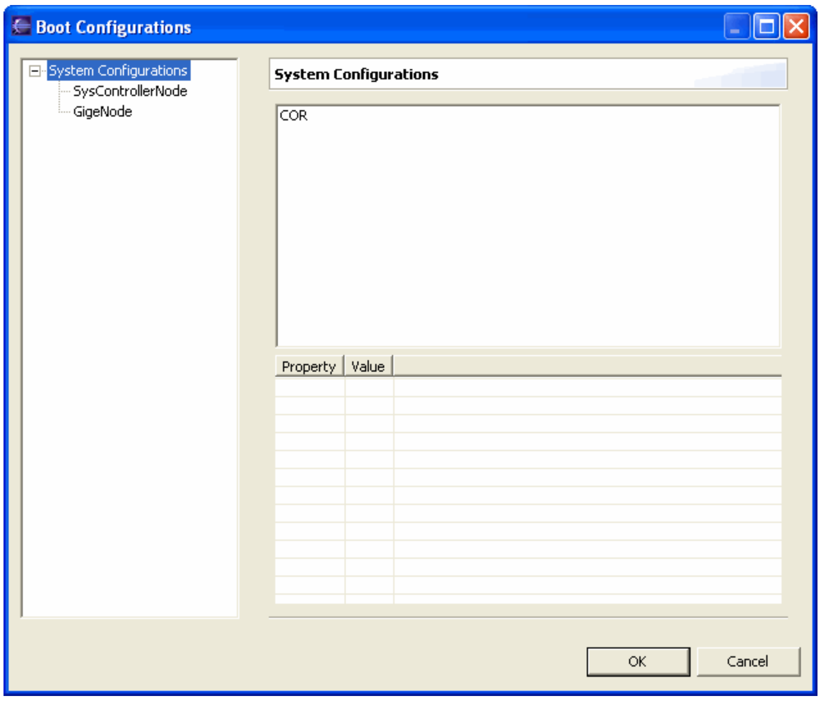
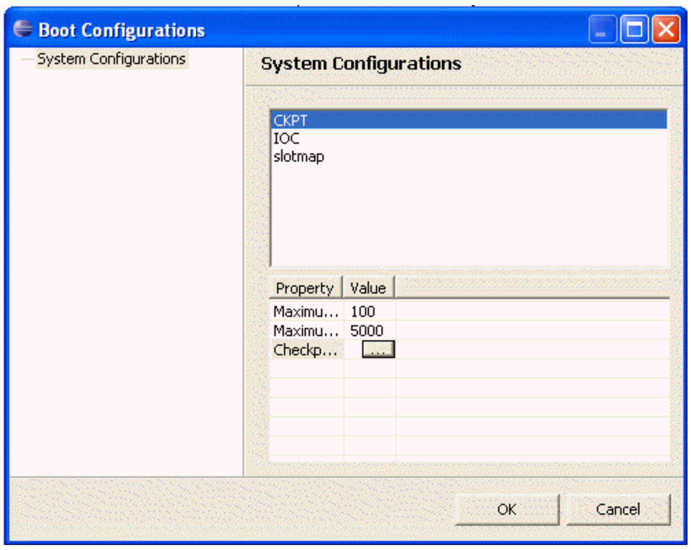


Figure 4-1 Boot Time Configuration for COR

The Boot Time dialog screen is divided into three frames. The left hand frame has System Configurations under which the nodes are present. The right hand frame is divided into two frames and has SISP component COR that can be configured appropriately.

The other SISP components that can be configured during boot time are also displayed under the SystemController Node and GigeNode. This is illustrated in Figure 4-2 .

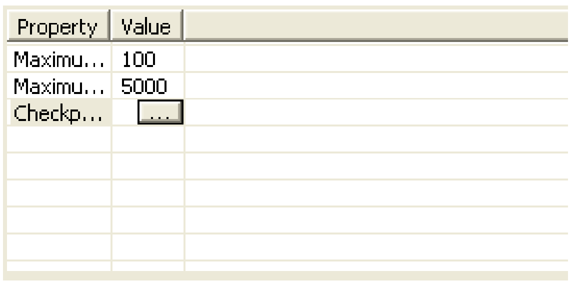


Property	Value
Maximu...	100
Maximu...	5000
Checkp...	...

Figure 4-2 Boot Time Configuration for Nodes

Contents - Index - Back

You can configure properties for SISP components during boot time. To configure properties, select the respective component and then specify their attributes and values. For example, to specify the attributes of CKPT, select it in the Boot Configurations dialog and specify its properties in the lower frame of the dialog box, as shown in Figure 4-3 .



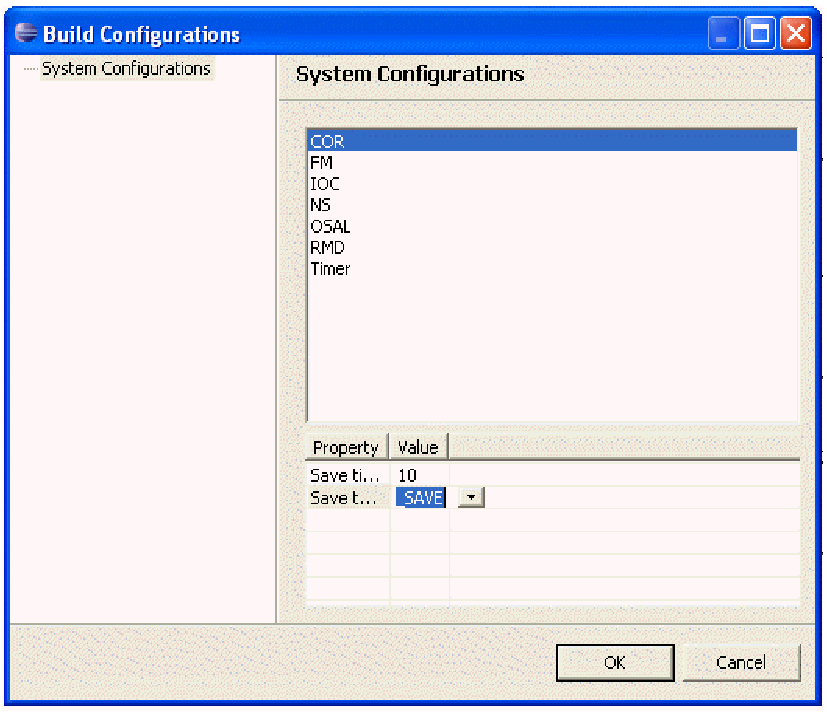
Property	Value
Maximu...	100
Maximu...	5000
Checkp...	...

Figure 4-3 Properties Dialog

Configuring Build-time Parameters

To configure Build time parameters, do the following:

Right-click from the Clovis Workspace and select ‘Build Configurations’. The ‘Build Configurations’ dialog appears, as shown in Figure 4-4 .



Property	Value
Save ti...	10
Save t...	SAVE

Figure 4-4 Build Time Configuration

The Build Time dialog screen is divided into three frames. The left hand frame has the node names, the right hand frame has SISP components names while the lower frame

Contents - Index - Back

provides options to specify the parameters. You can configure properties for about seven SISP components.

4.2 Configuring Boot Time SISP Components

You can configure the properties of three SISP components during boot time. They are:

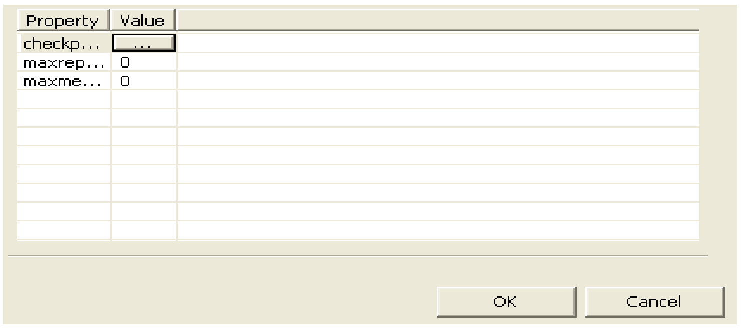
- Checkpoint Service (CKPT)
- Intelligent Object Communication (IOC)
- Slotmap

These are described in detail in the next section.

4.2.1 Checkpoint Service (CKPT)

The Clovis Checkpointing Service is an availability infrastructure component that enables applications to manage their state for redundancy purposes. As part of availability service, such as recovery from a failure, applications can resume their services from a saved state or checkpoint prior to the failure. This increases the availability of the applications by providing a seamless failover or restart.

CKPT consists of the following configurable properties during Boot time, which is shown in Figure 4-5 .

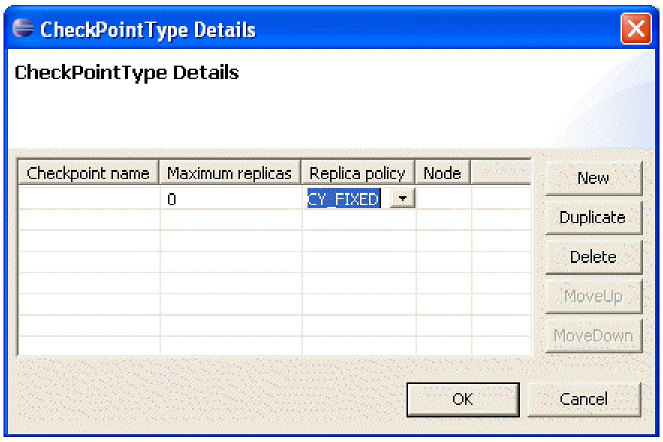


Property	Value
checkp...	...
maxrep...	0
maxme...	0

Figure 4-5 CPS Properties Dialog

Property	Description
Checkpoint	Checkpoint details
Maxreplicas	Maximum number of checkpoint copies to be made available
Maxmemory	Maximum memory that can be used for

To configure the Checkpoint details, click the button in the value field. The Checkpoint details dialog appears:



Checkpoint name	Maximum replicas	Replica policy	Node
	0	CY_FIXED	

Figure 4-6 Checkpoint Details Dialog

Property	Description
Checkpoint id	The ID of the checkpoint
Maxreplicas	Maximum number of copies for this checkpoint
Replica Policy	Node selection policy for a replica
Node	Node type details

While configuring the Checkpoint details, you will be required to configure the Node details also by clicking the button ... . The Node Type Details dialog appears, as shown in Figure 4-7

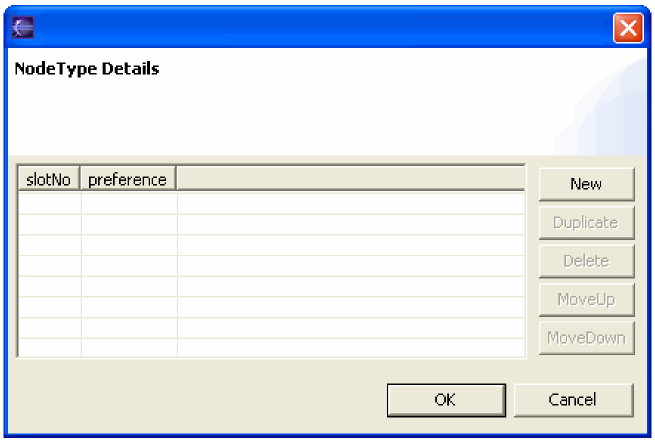


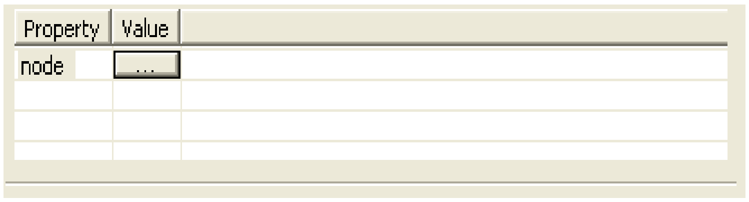
Figure 4-7 Checkpoint Node Details

Property	Description
Slot No.	The slot number for the Node
Preference	The slot preference for the Node

4.2.2 Intelligent Object Communication (IOC)

The Clovis IOC provides communication between various Clovis objects. It provides transport and OS neutral communication between Clovis objects. IOC component is designed to be modular to satisfy varying requirements, by allowing each system to include only the functionality needed by the system. IOC supports *best-effort* communication mode. IOC makes use of transport links to send and receive messages. Multiple user transports can be plugged into IOC.

IOC consists of the following configurable properties during Boot time, which is shown in Figure 4-8 .



Property	Value
node	...

Figure 4-8 IOC Properties Dialog

Property	Description
Node	One instance of an IOC

To configure the Node type details for each instance of IOC, click the button in the value field. The Node Type details dialog appears:

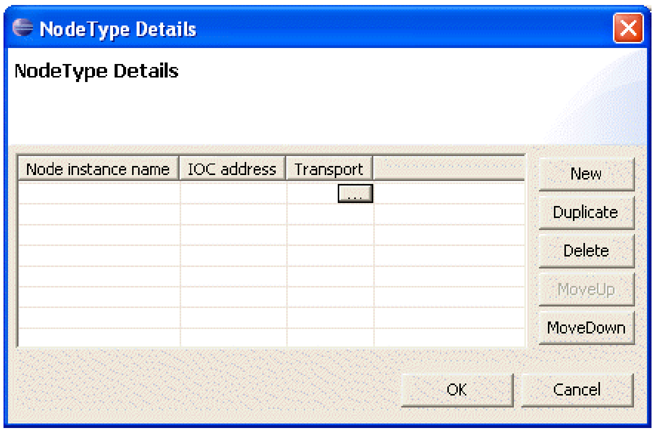
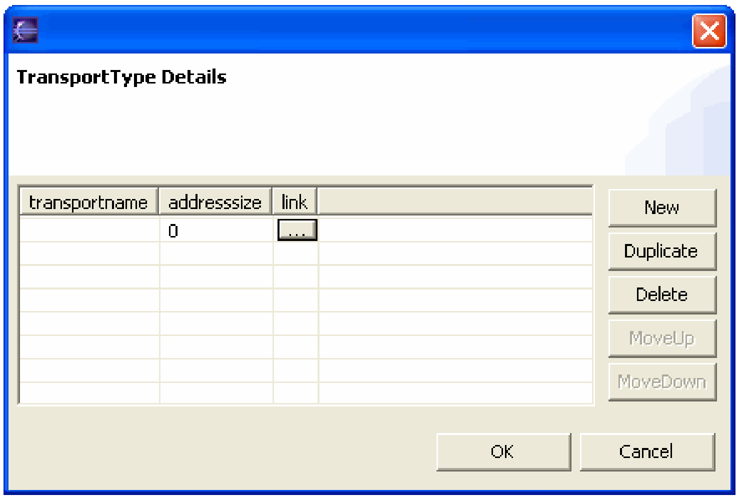


Figure 4-9 IOC Node Details Dialog

Property	Description
NodeInstanceName	Name of the node instance
IOCaddress	IOC address of the node

Once you are done with the configuration in NodeType details dialog, you must configure the Transport details. Click the three dots button to launch the TransportTypeDetails dialog, as shown in Figure 4-10 .

Contents - Index - Back

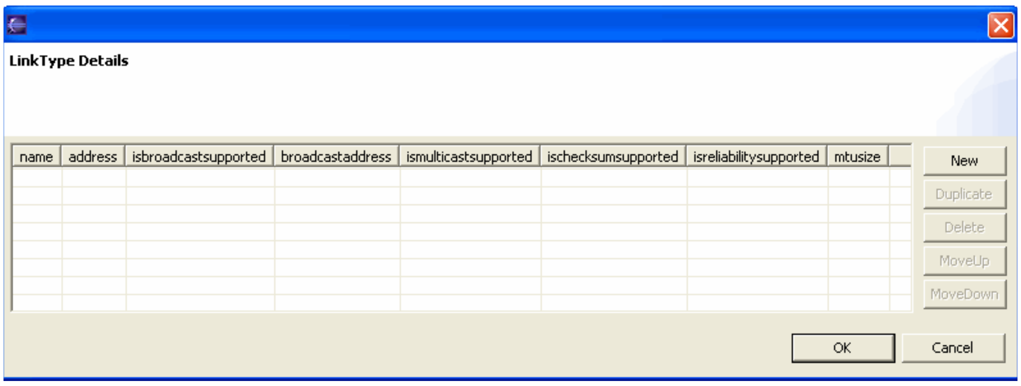


transportname	addresssize	link
	0	...

Figure 4-10 IOC Transport Details Dialog

	Description
Transport Name	Name of the transport. Currently UDP transport is supported by SISP. The customer can define own transport (Refer to CGAN010-Writing Custom IOC Transport Objects).
Address size	Size of the transport address. In the case of UDP Tranport, it is the size of the string that defines the link address. See Address for more information in the following page in LinkType Details section.
Link	Link details

After configuring the Transport details, you must configure the Link details. To configure the Link details, click the ... in the transport field. The Link details dialog appears, as shown in Figure 4-11 .



name	address	isbroadcastsupported	broadcastaddress	ismulticastsupported	ischecksumsupported	isreliabilitysupported	mtusize

Figure 4-11 IOC Link Details

Contents - Index - Back

Property	Description
Name	Name of the link
Address	Address of the link. For UDP transport, it is the IP address and port number. For example, 192.168.30.45:888
isbroadcastsupported	If broadcast is supported on the link
broadcastaddress	Broadcast address of the link
ismulticastsupported	If multicast is supported on the link
ischecksumsupported	If checksum is supported by the link
isreliabilitysupported	If reliability is supported by the link
mtusize	Maximum transmission unit size of the link

Contents - Index - Back

4.3 Configuring Build Time SISP Components

You can configure the properties of seven SISP components during build time. They are:

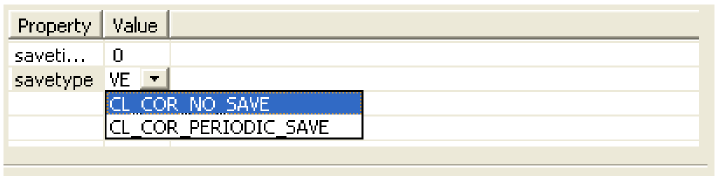
- Clovis Object Registry (COR)
- Fault Manager (FM)
- Intelligent Object Communication (IOC)
- NS
- OSAL
- RMD
- Timer

These are described in detail in the next section.

4.3.1 Clovis Object Registry (COR)

The Clovis Object Registry (COR) is used within the Clovis runtime environment and consists of a repository of Managed Objects (MOs) and Managed Service Objects (MSOs). COR provides an interface to manage the Managed Objects and Managed Service Objects within it. COR is a distributed, hierarchical, and object oriented database which captures the hierarchical relationship and attributes of different hardware and logical components of the system.

COR consists of the following configurable properties during build time, which is shown in Figure 4-12 .



Property	Value
saveti...	0
savetype	VE

Figure 4-12 COR Build Time Property Dialog

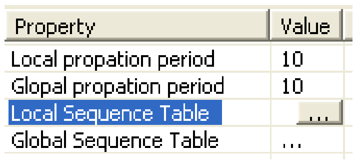
Property	Description
Savetime	Time in milliseconds for saving COR data in persistent storage
Savetype (CL_COR_NO_SAVE)	Don’t save
Savetype (CL_COR_PERIODIC_SAVE)	Save periodically. The frequency is specified by save time.

Contents - Index - Back

4.3.2 Fault Manager (FM)

The Clovis FaultManager (FM) is an EO that processes all application-related fault events. The FaultManager executes the recovery handler upon fault occurrence. The basic level of repair that an FM carries out is restarting the component. As part of its progressive fault repair policy, it will try to repair the faulty component first at the component level and if everything else fails, it will escalate the fault to a higher component which will reboot the node.

The FaultManager consists of the following configurable properties during build time, which is shown in Figure 4-13 .



Property	Value
Local propation period	10
Glopal propation period	10
Local Sequence Table	...
Global Sequence Table	...

Figure 4-13 FM Property Dialog

Property	Description
Local probation period	Store faults for FM local for the specified amount of time. If the fault takes place within the specified time, progressive repair actions will be taken.
Global probation period	Store faults for FM global for the specified amount of time. If the fault takes place within the specified time, progressive repair actions will be taken.
Local sequence table	Sequence table entry type details
Global sequence table	Sequence table entry type details

The following table specifies the Sequence table entry type details:

Property	Description
Category	Category of the fault
Subcategory	Sub category of the fault
Severity	Severity of the fault
Escalation Level	Levels of escalation for fault repairs
Function	Function taking care of fault repair handling

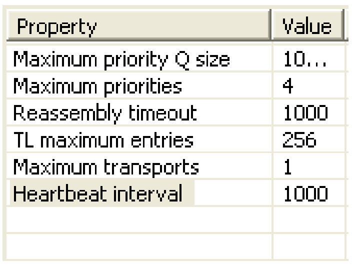
Contents - Index - Back

4.3.3 Intelligent Object Communication (IOC)

	Description
seqtablesize	Progressive repair table
Probation period	Time till which FM will wait for the progressive repair

The Clovis IOC provides communication between various Clovis objects. It provides transport and OS neutral communication between Clovis objects. IOC component is designed to be modular to satisfy varying requirements, by allowing each system to include only the functionality needed by the system. IOC supports *best-effort* communication mode. IOC makes use of transport links to send and receive messages. Multiple user transports can be plugged into IOC.

IOC consists of the following configurable properties during build time, which is shown in Figure 4-14 .



Property	Value
Maximum priority Q size	10...
Maximum priorities	4
Reassembly timeout	1000
TL maximum entries	256
Maximum transports	1
Heartbeat interval	1000

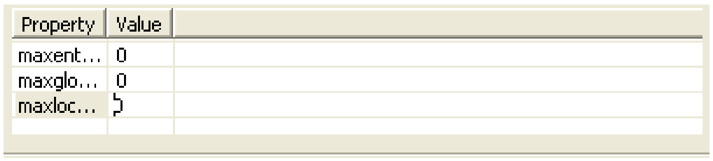
Figure 4-14 IOC Build Time Property Dialog

Property	Description
Maximum priority Q size	Maximum size of the priority queue
Maximum Priorities	Maximum level of priorities supported by IOC
Reassembly Timeout	Reassembly time interval of a fragmented packet in the kernel
TL Maximum entries	Maximum number of entries in a transparency layer
Maximum Transports	Maximum number of transports
heartbeatinterval	Time interval between two heartbeats

4.3.4 NS

Naming Service (NS) is a mechanism that allows an object to be referred by its 'friendly' name rather than its 'object (service) reference'. The 'object reference' can be logical address, resource id, and so on. Object reference is an opaque name so far as Name Service is concerned. Naming service returns the logical address specified by the 'friendly' name. Hence, Name Service helps in providing location transparency. This 'friendly' name is topology and location agnostic.

NS consists of the following configurable properties during build time, which is shown in Figure 4-15 .



Property	Value
maxent...	0
maxglo...	0
maxloc...	5

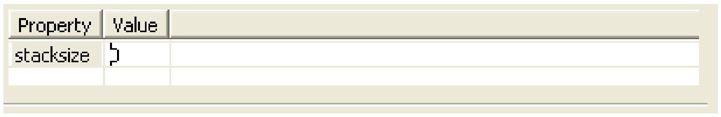
Figure 4-15 NS Property Dialog

Property	Description
Maximum entries	Maximum number of entries per context
Maximum global contexts	Maximum number of user-defined global contexts
Maximum local contexts	Maximum number of nodes local to user contexts

4.3.5 OSAL

The Clovis OS Abstraction Layer (OSAL) is a thin layer of software that provides a uniform set of interfaces into the underlying Operating System services. OSAL resides between the SISP and any other RTOS. Such RTOS can be either a commercial off-the-shelf or an internally developed operating system. This component allows Clovis components to remain the same irrespective of the underlying operating system. OSAL translates all the Clovis OS calls into the native OS calls.

OSAL consists of the following configurable properties during build time, which is shown in Figure 4-16 .



Property	Value
stacksize	5

Figure 4-16 OSAL Property Dialog

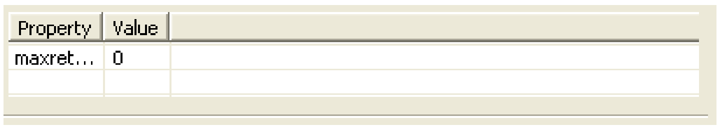
Property	Description
Stack size for task	It is an integer type argument you can use to define the stack size for a task. Each task has its own stack. The minimum and default size of the stack is 65536 while the maximum can be any positive integer value.

4.3.6 RMD

The Clovis Remote Method Despatch (RMD) invokes the function requested by the process, or the task running in the context other than the caller. RMD supports both the synchronous and asynchronous invocation of the function. RMD provides flexibility for you to invoke the function either by specifying its own byte buffer and length, or by creating the message and dispatching it. In a similar way it can pass on the pointer to the output buffer or the output message handle too to receive the reply.

The application creates an Execution Object (EO) to invoke and to export the functions. An application needs to export all its functions using EO to allow others to invoke them using RMD. Similarly an application should be an EO to invoke the function using RMD.

RMD consists of the following configurable properties during build time, which is shown in Figure 4-17 .



Property	Value
maxret...	0

Figure 4-17 RMD Property Dialog

Property	Description
Maxretries	Maximum number of retries for an RMD call

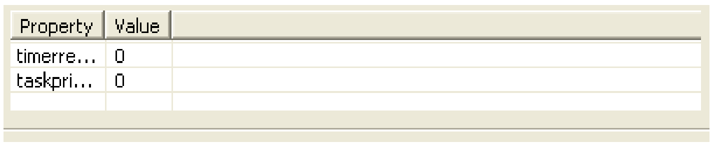
Contents - Index - Back

4.3.7 Timer

The Clovis Timer Library enables you to create multiple timers that execute a user-defined function when the timer expires. Timers are of the following two types:

- *One-shot Timers*—One-shot timers do not start automatically, once they are expired.
- *Repetitive Timers*—Repetitive timers are automatically restarted by the timer-library every time they expire.

The Timer Library depends on OSAL for creation of threads and memory allocation routines. ClovisWorks allows you to configure the Timer properties, shown in Figure 4-18 .



Property	Value
timerre...	0
taskpri...	0

Figure 4-18 Timer Property Dialog

Property	Description
Timer Resolution	Here you can define the timer resolution in milliseconds. The default value is 10 milliseconds and the maximum can be any positive integer value.
Timer Task Priority	Here you can define the priority of timer task. The default value is 150. It can have any integer value less than equal to 160.

Chapter 5
Generating Code and Testing

5.1 Generating Code

To generate code for the modeled NE, select ***Build Project*** option from the *Project* menu. The source code for the selected project will be generated. You can generate code for all the projects listed in the Clovis Workspace by selecting the ***Build All*** option. All the files required for SISP consumption are generated in a folder

Important!: Before generating code for your project using the ***Build All*** option, make sure the project is selected by clicking on the project in the Clovis Workspace.

Table 5-1 Generated Files

Generated Files	Associated .CS file	Directory Location	Description
clCorCfg.h	buildconfigs.py	<SISP_Location>/models/<project_name>/configs	This file captures compile time configuration parameters for COR. It contains information such as COR database file name, COR database save type and so on.
clFaultCfg.h	buildconfigs.py	<SISP_Location>/models/<project_name>/configs	This file captures design time progressive repair handler table for fault manager running on controller card.
clfmlCfg.c	fm.py	<SISP_Location>/models/<project_name>/configs	This file captures design time progressive repair handler table for fault manager running on blades.
clfmlRepairHandlers.c	fm.py	<SISP_Location>/models/<project_name>/configs	This file contains the implementation of user-defined progressive repair functions for fault manager running on blades.

Table 5-1 Generated Files

Generated Files	Associated .CS file	Directory Location	Description
clIocCfg.h	buildconfigs.py	<SISP_Location>/models/<project_name>/configs	This file captures build time configuration parameters like node-level heartbeating interval, maximum priority queue size, reassembly, time interval, maximum number of priorities, and so on.
clNsCfg.h	buildconfigs.py	<SISP_Location>/models/<project_name>/configs	This file captures compile time configuration parameters for NS. It contains information such as maximum number of entries per context, maximum number of user-defined global contexts and maximum number of local contexts.
clOsalCfg.h	buildconfigs.py	<SISP_Location>/models/<project_name>/configs	This file captures the build time configuration parameter such as thread stack size limit.
clRmdCfg.h	buildconfigs.py	<SISP_Location>/models/<project_name>/configs	This file captures build time configuration details for RMD. It has information such as maximum number of retries.
clTimerCfg.h	buildconfigs.py	/<SISP_Location>/models/<project_name>/configs	This file captures the build time configuration parameter such as task priority of the timer and timer resolution.
corMetaStruct.c	cor.py	<SISP_Location>/models/<project_name>/configs	This file contains the output of the Information model. The output of IM that is contained in CorMetaStruct.c is the cor class definition and MOTree definition. All the logical and physical MO trees are captured here.
corMetaStruct.h	cor.py	<SISP_Location>/models/<project_name>/configs	This file captures compile time configuration such as enum for COR class Ids, enum for attribute Ids, and so on.
omClassId.h	om.py	<SISP_Location>/models/<project_name>/configs	This file captures compile time configuration such as enum for OM class Ids.
<Component Name>alarmMetaStruct.c	alarmprofiles.py	<SISP_Location>/models/<project_name>/configs	This file is generated for each component and it contains information about modeling time alarm configuration for all the resources for each component.

Contents - Index - Back

Table 5-1 Generated Files

Generated Files	Associated .CS file	Directory Location	Description
<Component Name>AppMain.c	AppMain.py	<SISP_Location>/models/<project_name>/configs	This file is generated per component during build time and contains EO/component-related function call back stubs/implementation.
<Component Name>OAMPConfig.c	om.py	<SISP_Location>/models/<project_name>/configs	This file captures modeling time configuration information using ClovisWorks. It contains OM class definition and its corresponding function prototypes for components which use Alarm or Provisioning.
<Component Name>OAMPConfig.h	om.py	<SISP_Location>/models/<project_name>/configs	This file captures modeling time configuration information using ClovisWorks. It contains OM class definition and its corresponding function prototypes for components which use Alarm or Provisioning.
<Node Name>ckpt.xml		<SISP_Location>/models/<project_name>/configs	This file captures boot time configuration for the list of checkpoints in the system. It contains information such as names of the checkpoints and the nodes on which checkpoints can be present.
<Node Name>ioc.xml		<SISP_Location>/models/<project_name>/configs	This file captures the boot time configurations for IOC and its transport object. It has information like node name, node ioc address, link configuration parameters and so on.
<Node Instance Name>CpmConfig.xml		<SISP_Location>/models/<project_name>/configs	This file is generated per node during boot time and contains all the component, SU, and boot-related information.