Conventions

The following conventions are used throughout this document to help organize the presentation and communicate special semantics associated with portions of the text:

Table 1: Typographical Conventions

<table>
<thead>
<tr>
<th>Typographical Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courier font</td>
<td>Actual program, function, or variable names, or other text designed to be entered literally, as-is, to a program input stream; or which is directed as-is to the output stream</td>
</tr>
<tr>
<td>Italic font</td>
<td>Text that is a placeholder for a specific substitution that must be made to a program input stream</td>
</tr>
<tr>
<td>Courier, indented, blue</td>
<td>Console output, input examples, syntax prototypes (typically referenced by the preceding &quot;normal style&quot; paragraph)</td>
</tr>
<tr>
<td>[OPTIONAL-CONTENT]</td>
<td>Optional items in a syntax prototype are enclosed in square brackets</td>
</tr>
</tbody>
</table>

Table 2: Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OA</td>
<td>OpenAccess</td>
</tr>
<tr>
<td>OA class name sans &quot;oa&quot; prefix</td>
<td>an instantiation of the corresponding OA class. Examples: Design, Object, ScalarNet, Term, BitTerm, DMAttr.</td>
</tr>
<tr>
<td>oad</td>
<td>the main entry point in the Si2OADebug application interface</td>
</tr>
<tr>
<td>RTM</td>
<td>run-time model – A structured representation of the complete set of OA Objects, attributes, and relationships, associated with an application process at a given point during its execution.</td>
</tr>
<tr>
<td>Si2</td>
<td>Silicon Integration Initiative</td>
</tr>
<tr>
<td>WG</td>
<td>working group</td>
</tr>
</tbody>
</table>
Acknowledgements

The following members of the OpenAccess Debug Working Group (the "Debug WG" or just "WG") ([http://openeda.si2.org/projects/si2oadebug](http://openeda.si2.org/projects/si2oadebug)), in the course of their mission to investigate techniques to simplify the debugging of problems encountered using the OpenAccess Reference Implementation, applications linked with it, or OA data, have provided guidance, feedback, ideas, and promotion support for the development of this code at Si2:

<table>
<thead>
<tr>
<th>Name</th>
<th>Affiliation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johannes Grad</td>
<td>Cadence</td>
</tr>
<tr>
<td>Kevin Grant</td>
<td>Freescale</td>
</tr>
<tr>
<td>Shaun McEvoy</td>
<td>Cadence</td>
</tr>
<tr>
<td>John McGehee</td>
<td>Voom</td>
</tr>
<tr>
<td>Steve Potter</td>
<td>Silicon Navigator</td>
</tr>
<tr>
<td>Joanne Schell</td>
<td>Freescale</td>
</tr>
<tr>
<td>Sidney Wang</td>
<td>Synopsys</td>
</tr>
<tr>
<td>Jim Wilmore (Chairperson)</td>
<td>Intel</td>
</tr>
</tbody>
</table>

Special thanks are due Susan Carver (Si2, Inc.) for tireless dedication to revealing bugs in the code and creation of all the Flash demos.
CONTENTS

Conventions .................................................................................................................. 3
Acknowledgements ......................................................................................................... 4
CONTENTS ..................................................................................................................... 5
1 Overview .................................................................................................................... 8
   1.1 Prerequisites ......................................................................................................... 8
   1.2 Motivation .......................................................................................................... 8
      1.2.1 Opaque Handles ............................................................................................ 9
      1.2.2 Return by Value .......................................................................................... 9
      1.2.3 One-to-Many Relationships ....................................................................... 9
      1.2.4 Readability ................................................................................................. 10
   1.3 Implementation Approach .................................................................................. 10
      1.3.1 Interactive Display Output ....................................................................... 10
      1.3.2 Operational Summary .............................................................................. 11
      1.3.3 Traversal Details ....................................................................................... 12
      1.3.4 Dynamic RTM Elaboration ....................................................................... 12
2 Quick Start ................................................................................................................. 14
   2.1 Tested Platforms ................................................................................................. 14
      2.1.1 Console Output .......................................................................................... 14
      2.1.2 GUI Output ............................................................................................... 14
   2.2 Unpack the Release Tarballs ............................................................................. 15
      2.2.1 Si2OADebug Code ................................................................................... 15
      2.2.2 XSLT processor ......................................................................................... 15
   2.3 Check Top Level Contents .................................................................................. 16
   2.4 Edit Environment Information .......................................................................... 16
      2.4.1 Win XP Version ......................................................................................... 16
      2.4.2 UNIX Version ............................................................................................. 17
      2.4.3 226 (DM3) ................................................................................................. 17
   2.5 Edit the .si2oadoptions File ............................................................................. 17
   2.6 Test the Stand-Alone si2dis Utility ................................................................... 18
      2.6.1 Make and Run the Code .......................................................................... 18
      2.6.2 Try Other Testdata ..................................................................................... 19
      2.6.3 Try the GUI ............................................................................................... 19
   2.7 Debug a Fully-Linked executable ...................................................................... 19
3 Use Cases .................................................................................................................. 20
   3.1 Visualization ...................................................................................................... 20
   3.2 Debugging .......................................................................................................... 20
      3.2.1 Stand-alone Display of Persistent Data ...................................................... 20
      3.2.2 Interactive Display of Run-Time Data ........................................................ 21
      3.2.3 Embedded oad() Calls ............................................................................ 21
      3.2.4 Event Tracing ............................................................................................ 22
   3.3 Post-Processing .................................................................................................. 22
      3.3.1 Regression testing ..................................................................................... 22
      3.3.2 Analysis ...................................................................................................... 22
      3.3.3 Reformatting ............................................................................................. 22
4 Interface .................................................................................................................... 23
   4.1 Data Traversal and Display Functions ............................................................... 23
1 Overview

Si2OADebug is a utility that facilitates display of OpenAccess Objects and their attributes and relationships to other Objects. It can be used

- **Stand-alone** to display the contents of a persistent Design on disk.
- **Interactively** to display run-time OA data using program code variable names or addresses inside a debugger (dbx, gdb) session that is executing application.
- **Embedded** in an application to export any parts of the run-time model (RTM) for later analysis.

Results can be configured to display to a

- text-based output stream
- GUI based viewer with "dynamic" HTML

Makefiles help automate build and setup functions. Section 2 instructions provide a quick start for unpacking and installing the code.

1.1 Prerequisites

Effective use of Si2OADebug and this manual requires some knowledge of the OA model; otherwise, the inputs required and output generated will be difficult to understand.

- The output is frequently abbreviated. For example
  
  - The words, "ConstraintGroup", are displayed as, "CG"; so, the attribute indicating an Object has a default ConstraintGroup is rendered as, `hasDefaultCG`, differentiating it from `hasObjectCG`, which describes one of the other types of ConstraintGroup Booleans.
  
  - The CurrentDensity Layer attributes `peakACCurrentDensity` and `avgACCurrentDensity` are abbreviated as `peakAC` and `avgAC`, respectively.

- A Design actually has three separate Lib, Cell, and View name attributes; however, they are displayed together, with path separator characters, as the value of a single "lcv" attribute.

Occasional reference to OA API documentation will assist in mapping display artifacts to specific details relating to attribute semantics and the Object hierarchy.

1.2 Motivation

The very object-oriented design features that facilitate use of the OpenAccess API can really get in the way of debugging the CAD data it so carefully encapsulates.
1.2.1 Opaque Handles

Instead of public member variables, OA classes hide attribute and relationship information behind methods. This is a useful interface design strategy but makes accessing information about OA Objects inside a debugger rather awkward.

For example, requesting gdb to display a ModNet yields nothing except an "opaque" Object handle. While it is technically a pointer to an object instantiation, dereferencing it in the debugger produces little useful information:

```
Breakpoint 2, main (argc=4, argv=0xbfffcc84) at testcase.cpp:124
124 cout << "got ModNet" << modNet << endl;
(gdb) p modNet
$1 = (oaModNet *) 0xb75d0612
(gdb) p *modNet
$2 = { <oaModObject> = {<oaDesignObject> = {<oaObject> = {id = 4 '\004'}, <No data fields>}, <No data fields>}, <No data fields> }
```

Accessing attribute information requires invocation of individual Object member functions one at a time. This quickly becomes tedious with interspersed console dialog that makes the output hard to read:

```
(gdb) p modNet->getSigType()
$3 = {value = oacSignalSigType, static names = 0x80ab1d8}
(gdb) call modNet->isImplicit()
$4 = 0
(gdb) call modNet->isEmpty()
$5 = 0
...
```

1.2.2 Return by Value

Many attributes are not return values of a function but are instead stored in one (or more!) of the function arguments, which must have been allocated by the caller and passed in by reference. This makes accessing such data complicated − even impossible − if there is no variable of the right type available in the source code (since most debuggers do not allow construction of temporary variables with user-defined class types). For example,

```
(gdb) call block->getBBox(box)
(gdb) p box
$6 = {lowerLeftVal = {xVal = -290, yVal = -187},
      upperRightVal = {xVal = 707, yVal = 880}}
(gdb) call modNet->getName(ns,str)
No symbol "str" in current context.
```

1.2.3 One-to-Many Relationships

Displaying one-to-many relationships (such as the Shapes in an LPPHeader, the InstTerms connected to a Net or the Terms it owns) is important for understanding how parts of a design relate to each other. However, inside a debugger such information would require many function calls: to obtain Collections of Objects, iterate over them, perhaps traverse to each one for a name attribute, etc. Moreover, bare
handle references without a corresponding compile-time typed variable would need individual, explicit type-casting to be interpretable to the debugger. This is just too much work for the real-time console.

1.2.4 Readability

What Object information can be produced by such brute force methods ends up fragmented with hit-or-miss formatting (including user-fumbled inputs) scattered among multiple prompts on the console. A more coherently organized view of such data would significantly aid visualization of the design.

1.3 Implementation Approach

Si2OADebug addresses the obstacles to easy access of model information by providing the algorithms for complete traversal of OA model Objects and mechanisms for displaying those Objects, their attributes, and associations to other Objects, generating data that can be displayed in a web browser as shown in Figure 1.

1.3.1 Interactive Display Output

The makefile setup details makes oad symbols available to the debugger (8), which can be run on the target application in the usual way. Breakpoints can be set at which, in addition to normal debugger analysis commands, the Si2OADebug interface functions, all overloaded versions of oad(), can be invoked to display various portions of the RTM.

Figure 1: Browser Display of OA Model Data
The resulting XML output can be directed in-line to the console or to a browser. On the console, the raw XML will be displayed. When sent to a browser, however, an XSL transform will be applied to it (either via stand-alone XSLT processor or by the browser's native XSL capability), turning it into HTML with Javascript enhancements to provide various levels of active content. Figure 2 illustrates the interactions between the debugger and called oad functions, and the outputs generated and processes spawned as a result.

A set of options (4.1.4) controls various aspects of the processing and display.

### 1.3.2 Operational Summary

When the debugger calls one of the oad() "display" entry points (as opposed to those that manipulate option settings), the traversal engine in si2oad.cpp starts at the supplied root object (or the entire RTM if no arguments are given) and performs a systematic, depth first traversal of the model. The si2oadutil.cpp module processes utility object data. Each object processed consists of some number of associations. Processing is sorted into two groups:

- Association data that can be expressed as a simple name=value pair. These associations are all arranged together at the start of the containing object.
- Data that has several such pairs, each of which must be associated together in a separate grouping.

#### 1.3.2.1 Stand-alone

If the output target option is set to "console" then the resulting XML is sent to stdout.

#### 1.3.2.2 Interactive

If a GUI (browser) is the target, then the output is sent to a file in the WWW server "root" directory that is set in the options (4.1.4). If an XSL processor was selected in the options, a system() call runs that tool to create an HTML file in that same server root directory. Then,
A web "nanoserver" (si2oadwww.cpp) is started by creating a socket bound to a free port (selected by the OS) and then looping to accept connections on that port.

♦ The accept loop is necessary both to handle the multiple file components (images, Javascript, etc.) on the main page, as well as subsequent requests for new XML traversals (as a result of clicks on the IDREF attribute of associations that reference objects not included on the data in the current page).

♦ This internal web server's call to the accept() function blocks on the socket until a special "quit" request is made by the browser. This is normally accomplished by clicking a special button that displays at the top of the HTML produced by the XSL translation. However, if all browser instances were closed, or if the raw XML was being displayed, no such button will exist. In such a case, a browser can be started manually to display the special localhost page named "quit". When the quit request is received, the socket and port are closed.

The browser set in the options is spawned via system() call and handed the URL of localhost plus the selected port, and the HTML file that was created (or the XML file if the options are set to use the browser's internal XSL processing, or if the external XSL processor failed).

1.3.3 Traversal Details

Many OA calls have side-effects that can change the RTM or persistent store. Traversal under OAD is intended to be non-destructive to both these models. Side-effects are prevented entirely in Si2OADebug, or controlled by an option setting; for example,

• getConstraintGroup() is not called if !hasConstraintGroup()
• getMaster(), getMasterTerm(), getMasterModule() are not called if !bound()
• getMasterOccurrence() is not called if option expandocc is not set to t
• ParasiticNetworks (and partitions) are not loaded unless the loadnetworks option is set to t
• getBBox() will not be called unless dobbox() is set to t.

1.3.4 Dynamic RTM Elaboration

The interactive version of OAD has a bit more flexibility when relationships are encountered that reference Objects not currently open in the RTM. Assuming a relevant option (1.3.3) has not been set to cause automatic opening, an active “sentinel” of some kind is planted in the display output to provide a method for the user to cause the Object to open, if desired. These sentinels are represented as values in capital letters, as summarized by the following Table:

For example,

• Ref Objects display a master reference in an attribute group that includes the Lib, Cell, and View names, as well as a design attribute, which has either a

  ♦ handle, if the master is already open in the RTM
  ♦ sentinel – the keyword, BIND – if that master has not been bound yet

  Clicking on BIND will cause a callback that binds the master and displays it in a separate window.
• An unexpanded master Occurrence reference of an OccInst has the “value”, EXPANDOCC. Clicking on such a sentinel interactively will cause the Occurrence to be expanded and displayed in a separate window.

• If the Tech for a Lib is not already open at the point in the application code where OAD was invoked, then the Lib's tech attribute will show the sentinel, OPEN, instead of a handle to the Tech.

Each call that explicitly opens an Object that was not already in the RTM will add the handle of that Object to a list. When the oad() call is terminated and control returns to the command line, each Object on that list will be purged, unloaded (whatever is appropriate for that Type) to return the RTM to its original state before the oad call.

<table>
<thead>
<tr>
<th>Container Object</th>
<th>Attribute Group</th>
<th>Simple Attribute</th>
<th>Active Sentinel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lib</td>
<td></td>
<td>tech=</td>
<td>OPEN</td>
</tr>
<tr>
<td>Inst, ModDesignInst, OccInst</td>
<td>master</td>
<td>design=</td>
<td>BIND</td>
</tr>
<tr>
<td>ModModuleInst</td>
<td>master</td>
<td>module</td>
<td>BIND</td>
</tr>
<tr>
<td>InstHeader, ModInstHeader, OccInstHeader</td>
<td>master</td>
<td>design=</td>
<td>BIND</td>
</tr>
<tr>
<td>ModInstHeader, ModModuleInstHeader</td>
<td>master</td>
<td>module</td>
<td>BIND</td>
</tr>
<tr>
<td>CustomVia, CustomViaDef</td>
<td>master</td>
<td>design=</td>
<td>BIND</td>
</tr>
<tr>
<td>CustomViaHeader</td>
<td>master</td>
<td>design=</td>
<td>BIND</td>
</tr>
<tr>
<td>Via</td>
<td></td>
<td>viaDef=</td>
<td>BIND_VIASETDE</td>
</tr>
<tr>
<td>CellView</td>
<td>primary</td>
<td>design=</td>
<td>BIND</td>
</tr>
<tr>
<td>ParasiticNetwork, Partition</td>
<td>pn</td>
<td>pn</td>
<td>LOAD</td>
</tr>
<tr>
<td>OccInst, OccModuleInstHeader</td>
<td>master</td>
<td>masterOcc</td>
<td>EXPAND</td>
</tr>
<tr>
<td>InstPropDisplay, OccInstPropDisplay</td>
<td>masterProp</td>
<td>masterProp</td>
<td>BIND</td>
</tr>
<tr>
<td>TextOverride</td>
<td>masterText</td>
<td>masterText</td>
<td>BIND</td>
</tr>
</tbody>
</table>
2 Quick Start

An automated test script, `testrelease.sh`, can be used to test the release package. It illustrates running the tool with various make configurations and testcases. The output can then be compared against the file, `testrelease.gold`, to verify the code is working as expected. Since many details (including console prompts, dates, hexadecimal address values, etc.) will likely be different for various install configurations and platforms, an exact diff is probably not possible. But except for those details, a test run should surface any major problems with the release or its installation.

2.1 Tested Platforms

The batch version consists of straightforward C++ code and should build with any ANSI compiler. Si2oaDebug is built as a shared library.

The first versions of the code were built on Solaris 9. However, the lion's share of releases to date have been tested only on Linux with gdb, and WinXP with VisualStudio (.NET).

<table>
<thead>
<tr>
<th>OS</th>
<th>Compiler</th>
<th>Bits</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuSE 10</td>
<td>gcc 4.1.0</td>
<td>32</td>
<td>firefox 1.5.0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>firefox 2.0</td>
</tr>
<tr>
<td>SuSE 10.1</td>
<td>gcc 4.1.0</td>
<td>64</td>
<td>firefox 1.5.0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>firefox 2.0</td>
</tr>
<tr>
<td>RH EL 30</td>
<td>gcc 3.2.3</td>
<td>32</td>
<td>mozilla 1.4</td>
</tr>
<tr>
<td>RH EL 40</td>
<td>gcc 3.4.3</td>
<td>32</td>
<td>mozilla 1.7.7</td>
</tr>
<tr>
<td>XP</td>
<td>VisualStudio 2003</td>
<td>32</td>
<td>firefox 2.0</td>
</tr>
</tbody>
</table>

2.1.1 Console Output

If linked in and called from within an application, any ANSI linker should work. However, the makefiles are set up to insert that lib into an executable to enable its use with a debugger using `LD_PRELOAD`, which may not be available on all platforms.

2.1.2 GUI Output

Using the GUI (browser) to view the output pulls in a bunch of other dependencies:

- Some XSL transform code and a processor to map it to HTML
- A browser to display the resulting HTML resulting from the
A decent amount of Javascript code using several DOM features (notoriously supported with varying consistency across browser implementations)

There is a wide variation browser support at the level needed for this GUI display. Platform/browser combinations other than those in Table 5 may also work with varying degrees of success.

### 2.2 Unpack the Release Tarballs

#### 2.2.1 Si2OAODebug Code

The code release itself is in tar, gzip format, and includes the date it was packed in the filename. For example, to unpack the 1.1.4 release, the following commands would be used:

```
$ gunzip -c si2dis081706.tar.gz | tar xvf -
```

#### 2.2.2 XSLT processor

The native output format for Si2OAODebug is XML. If the output option (4.1.4) is set to "console" XML will be displayed on the console. However, setting that option to "gui" will transform that XML into HTML, adding active content to make it much more readable, especially as the volume of output gets larger. This is accomplished via XSLT from either

- the browser's native XSLT capability (a feature of Mozilla, Netscape, and other browsers)
- a stand-alone processor (recommended because many browser XSLTs do not work properly)

The xslttype option (4.1.4) selects which these two methods will be used, "processor" or "browser", respectively. If that option is "none" or if XSL transormation fails, the raw XML will be sent to the browser – which may still be better than console display since many browsers will add collapse/expand features to XML content automatically.

If the browser does not have XSLT capability, the Xalan processor (and xerces XML parser) can be used for the transformation (8). A self-contained package of these tools is available for download separately from the Si2 OpenAccess Debug project area of the [OpenEDA.org](http://www.openeda.org) site. The following log illustrates unpacking these tools for use.

```
$ gunzip -c XML.tar.gz | tar xvf -
XML/
XML/bin/
XML/bin/Xalan
XML/lib/
XML/lib/libxalan-c.so
XML/lib/libxalan-c.so.19
XML/lib/libxalan-c.so.19.0
XML/lib/libxalanMsg.so
XML/lib/libxalanMsg.so.19
XML/lib/libxalanMsg.so.19.0
XML/lib/libxerces-c.so
XML/lib/libxerces-c.so.26
XML/lib/libxerces-c.so.26.0
XML/lib/libxerces-depdom.so
```
2.3 Check Top Level Contents

Change directory to the top of the release tree and look at the contents.

```
$ cd si2dis110906
$ ls -F
  detpara/   makedefs.gcc*   makefile.defs* sample/   si2dis/
  makedefs.forte*  makedefs.HPaCC*  mdparies/  si2assert.h*  si2distest/
```

There should be

- several makefile components
- the main `si2dis` directory which houses both the
  - Si2OADebug library code
  - stand-alone `si2dis` executable useful for batch display of a persistent design
- the `si2distest` directory containing testcases to practice the use of Si2OADebug with linked executables that have various kinds of OA data
- 3 data directories
  - `sample/` (a basic set of data for testing)
  - `mdparies/` (a larger, slightly more realistic design)
  - `detpara/` (some detailed parasitics)

2.4 Edit Enviornment Information

2.4.1 Win XP Version

Read the README.Win32.txt file for information about building with VisualStudio. Two additional components may need to be installed to take advantage of the full capability of the tool. Both can be downloaded either from their source sites or from the OpenEDA “Redistributed Components” page:

2.4.1.1 RemoteLib

The insertDLL code is used to inject the oaD libs into a process (in place of the LD_PRELOAD technique used on UNIX platforms). If not explicitly linking the oaD libs in with the executable (meaning that exe has a call to an oaD symbol that needs resolution), then this library [or equivalent] must be downloaded and unpacked in the `si2dis/` directory.
2.4.1.2 GNU Regex

This GPL library implements the regular expression filter capabilities of the tool. If plans include use of the regex feature to filter output (whether using console or gui mode) then this library [or equivalent] must be downloaded and unpacked in the si2dis/ directory.

2.4.2 UNIX Version

Edit makefile.defs

Set key make variables to appropriate values in the target execution environment. In particular, the following paths almost certainly will need to be customized for each installation:

• PATH2DEFS=/absolute/path/to/makefile.defs
• ROOT_OA=/absolute/path/to/root/of/OA/install/ (i.e., where lib/ and bin/ are).

It is possible other changes may be needed near the top of that file for special situations. It may also be necessary to set the #include makedefs.* appropriate for the architecture and tools being used, as well as edit that file to account for other variations in the environment. On a 64 bit platform, the following setting is probably also necessary:

LIBDL = /lib64/libdl.so.2

The default setting create bin/ and lib/ directories in the package root.

DIR_BIN_OAD = $(DIR_BIN_LABS)
DIR_LIB_OAD = $(DIR_LIB_LABS)
DIR_OADFILES = $(DIR_LIB_OAD)/si2oadebug

2.4.3 226 (DM3)

Beginning with v1.4.2, support is available for both DM3 (226p052) and DM4 (p007) OA release versions, implemented in both

• code — This should not require manual editing. The oacDataModelRevNumber is checked via #ifdef, skipping code if its value is 4.

• makefiles — These will require manual editing. Search for “226” in both
  ♦ makefile.defs — To point paths to a 226 install and to activate the “D” suffix that was added to OA library names in DM3 releases.
  ♦ Makefiles in the si2dis/ and si2distest/ directories to point path settings to a 226 install.

2.5 Edit the .si2oadoptions File

The default option settings in the code are designed for limited console output only. Use of the GUI will require several environment-dependent settings. Copy an .si2oadoptions file into one of the following locations:
• the directory that will be the current working directory at the time the options are read
• $HOME (the second location tried in case no options file exists in the cwd)

Set key GUI options to conform to the current environment:

• **browser** – If mozilla is not a viable browser, or it is not in the PATH of the executing process, change this value accordingly (which may mean including an absolute path).

• **xsltype** – If the XML/ directory will not be installed, or if the browser is to be used to perform the XSL transformation from Si2oaDebug's XML output to HTML, then set this to "browser".

• **oadoc** – Set to the proper URL or absolute path (if locally installed) location of the OA doc/oa/html/ tree. This is used to link to the correct OA API class page when, using "gui" as the output target (option output), a click is made to an OA class name.

• **xslexe** – If option xsltype was set to "processor", set this to the correct absolute path of the XSL processor, which could be the XML/ directory available as a companion download (2.2.2).

• **xsllib** – If option xsltype was set to processor, set this to the correct absolute path of any shared objects needed by the XSL processor identified in option xslexe (2.2.2).

Other settings may also be required. Chapter 5 documents the available options, their default values, and semantics associated with them.

### 2.6 Test the Stand-Alone si2dis Utility

Try a simple test of the stand-alone si2dis to display a persistent design. Change to the si2dis/ directory

```
$ cd si2dis
$ make install
```

#### 2.6.1 Make and Run the Code

Use the run target to kick off the stand-alone si2dis tool that has the Si2OADebug libraries explicitly linked in and calls oad() directly inside the code. The testrelease.gold file included with the release shows several run examples and their outputs. The simplest (using command-line arguments specified by the INPUT make variable in the si2dis/Makefile)

```
$ make run
```

Note, in the default .si2oadoptions file, option output is set to "console", which will cause the output to appear on the console and be copied to si2dis.out which can be diff'd against the file, gold.sample.out. To avoid console output (which could be voluminous), the makefile.defs "run", "gdb", or "oad" targets can be edited to convert the pipe to the tee program into a simple redirect to an output file name. For large XML output, the tee can be overridden as follows:

```
$ make run TEE2FILE="" ...send to console only'
$ make run TEE2FILE=">file.txt" ...send to file.txt only'
```
2.6.2 Try Other Testdata

Run the same si2dis stand-alone tool on a larger library. First change option output in the .si2oadoptions to be "gui" (this would be an excessive amount of output to send to the console):

$ sed 's/^output .*/output gui/' .si2oadoptions >.si2oadoptions-gui
$ cp .si2oadoptions-gui .si2oadoptions

Create a lib.defs file because this design requires definitions for a Tech in a different Lib but only one Lib mapping can be specified on the command line for si2dis:

$ echo DEFINE techLib ../mdparies/LibDirTech >lib.defs
$ echo DEFINE mdparies ../mdparies/LibDir >>lib.defs

Now run the tool specifying the Lib, Cell, and View names to be displayed:

$ make run INPUT="deflib mdp_aries layout"

2.6.3 Try the GUI

Set the option, output=gui, and then run the tool. Make sure the browser option is set to an executable in the PATH (or includes an absolute path).

If XSL runs successfully (whether via xsl=processor or xsl=browser) to convert the XML to HTML, then there are many active sites on the screen. Unfortunately, to reduce HTML volume for performance reasons, few of these are marked individually with tags; hence, automatic change of the cursor to a “hand” via CSS is impossible. Becoming familiar with the kinds of things are active HTML will be necessary for effective use of the GUI.

2.7 Debug a Fully-Linked executable

While the si2dis/si2dis.cpp program has a call to an oad() function embedded in its source code, the si2distest directory has a couple of testcase "nano-apps" that illustrate running Si2OADebug with a fully-linked application. Although the Makefile in that directory actually builds a testcase from source, it is then linked with the oad code at run-time, showing how this can be accomplished with a commercial tool for which source (or even unlinked objects) may not be available.
3 Use Cases

3.1 Visualization

Both for designs stored on disk and for data changing dynamically in an application at run-time, a mechanism for visualization of the complete OA model represented by that data can help an engineer in a variety of ways.

3.2 Debugging

The original motivation driving creation of Si2OADebug was the desire facilitate display of a complete and coherent view of RTM data to assist in debugging application failures.

3.2.1 Stand-alone Display of Persistent Data

The si2dis/ directory houses a stand-alone display tool, si2dis, which requires "Lib Cell View" command line arguments representing the persistent design data to be displayed (resolving the LibPath from the lib.defs file). The tool can be invoked as follows to run the default input set in the Makefile:

```
make run
```

The Makefile automatically sets various environment values needed, as well as default arguments for a test design. However, these settings can be performed explicitly on the command line (or using other techniques) allowing si2dis to be run without using make, as in the following example:

```
LD_LIBRARY_PATH=.:/home/224/lib/linux_rhel21_32/dbg \
PATH=:$PATH:/home/224/bin \
SI2OADdocroot=/home/d/docroot \
si2dis INPUT-ARGS
```

The default arguments supplied by the Makefile can be changed easily by overriding the value of the INPUT make variable:

```
make run INPUT="..."
```

3.2.1.1 Command-Line Arguments

To display persistent design data requires specification of the Lib, Cell, and View names, plus an optional filesystem path to the Lib director, as follows:

```
make run INPUT="LIBNAME CELLNAME VIEWNAME [LIBPATH]"
```

If the character / is used for both CELLNAME and VIEWNAME, the whole Lib will be displayed. If there is a LIBPATH, then only the Lib in that path will be opened; otherwise, `oaLibDefList::openLibs()` will be executed (which will cause an error if no `lib.defs` file is located in one of the default search locations).
3.2.1.2 Interactive Mode

If the arguments provided are insufficient to uniquely specify a target Design, si2dis will enter interactive mode and query the user for this information.

3.2.1.3 Help

A simple reminder of the various command-line arguments can be produced on the console by running,

```
make run INPUT="-h[elp]"
```

3.2.1.4 Options Files

Used with the following arguments, si2dis will write the current option settings to the specified file and terminate.

```
make run INPUT="-w[rite] OPTIONS-FILENAME"
```

When running si2dis to display a persistent design, an –o argument may precede the design information, followed by the (relative or absolute) path of a valid oad options file. This file must be in the same format as a file created by the -w argument (as described above), though only the option names and values to be changed need be included in the file. When an options file is specified in this fashion, any values in it read will overwrite those set by any default .si2oadoptions that might exist.

```
make run INPUT="-o[ptions] OPTIONS-FILE LIB CELL VIEW LIBPATH"
```

Regular expression filtering on Object Type and name may be included (prior to the Lib name argument) using the –x option followed by a pathExpression (section 6):

```
make run INPUT="-x pathExpression LIB CELL VIEW"
```

3.2.2 Interactive Display of Run-Time Data

The Si2OADebug binaries can be used with any application and within a debugger (such as gdb or dbx) session. At any breakpoint any of the Si2OADebug interface functions can be invoked:

The testcase, test1.cpp, in the directory si2distest/ builds on the example design data in sample/ directory, creating enough oaObjects to provide a good walk-through of the display features. The testcase illustrates use of the Si2OADebug libraries with any application without having to touch the application source code.

A visual representation of the data being created by application code can help flatten the learning curve for engineers learning the OA model for the first time, as well as experienced OA hackers delving into some corner of the API for the first time.

3.2.3 Embedded oad() Calls

This utility and its Makefile illustrate how to link explicity with the Si2OADebug libraries to enable embedding oad() calls in any application code.
Any application can include the Si2OADebug interface header to enable use of all `oad()` interface functions directly.

```c
#include "si2oad.h"
```

Once an application is linked with Si2OADebug libraries, the `oad()` interface functions may be invoked directly by the application code to provide snapshots of any subset of the OA RTM. Such snapshots can then be post-processed for various purposes.

### 3.2.4 Event Tracing

The observer option (4.1.4) provides a hook for insertion of arbitrary user code into the OA RTM.

### 3.3 Post-Processing

The ASCII output format lends itself to automated post processing. Many mature commercial and free tools exist to manipulate XML – parsers, XSL transformers, editors, etc.

#### 3.3.1 Regression testing

A "golden" snapshot of the design data resulting from a particular test run of an application under testing can be saved for use with simple ASCII diff tools as the application evolves.

#### 3.3.2 Analysis

All attributes and relationships among data objects are available in the XML output. Unexpected artifacts in the output can aid in the detection of bugs in an application or errors in the data.

#### 3.3.3 Reformatting

The `si2oad.xsl` transform illustrates one reformatting use case. This XSLT converts the XML description of the design data into HTML, adding Javascript for active content features. The result can be displayed in a browser to enhance the visualization of the model.

XSL (or a program written in any language) could be used to filter or change the appearance the resulting output. Such a transformation could be performed dynamically as the output is generated.
## 4 Interface

From the command-line of a debugger session or within application code, two sets of interface functions are available for accessing data and changing execution options dynamically (Table 6).

- Overloaded versions of the single function name, `oad()`
- Uniquely-named functions – For debuggers (like VisualStudio) that have trouble calling overloaded functions at run-time, a non-overloaded equivalent function name is available (colored blue in Table 1).

### Table 6: Interface Functions

<table>
<thead>
<tr>
<th>Debugger call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oad()</code></td>
<td>Display the entire OA state, traversing to each <code>oaObject</code> (the Session, all open Libs, etc), showing all associations.</td>
</tr>
<tr>
<td><code>oadr()</code></td>
<td>Display the managed Object represented by the variable named, <code>variable</code>.</td>
</tr>
<tr>
<td><code>oad( variable )</code></td>
<td>Display the utility object represented by the variable named, <code>variable</code>. [NOT YET FINISHED]</td>
</tr>
<tr>
<td><code>oad( variable )</code></td>
<td>Display the utility object represented by the variable named, <code>variable</code>. [NOT YET FINISHED]</td>
</tr>
<tr>
<td><code>oad( 0xAddress )</code></td>
<td>Display the <code>oaObject</code> whose &quot;handle&quot; is <code>Address</code> (which must be preceded by the literal string &quot;0x&quot;)</td>
</tr>
<tr>
<td><code>oada( 0xAddress )</code></td>
<td>Display the OA state, traversing to each <code>oaObject</code> (the Session, all open Libs, etc), showing all associations.</td>
</tr>
</tbody>
</table>

### Example uses of the various options in Table 6 can be seen in the testrelease.gold file (2).

## 4.1 Data Traversal and Display Functions

### 4.1.1 Complete RTM

Calling `oad()` – or the non-overloaded synonym, `oadr()` – with no arguments will traverse and display the entire run-time model.
4.1.2 Managed oaObject

Passing the name of a variable in the code with a valid handle to a managed oaObject to oad() – or the non-overloaded synonym, oadm() – starts the traversal and display with that Object.

The following gdb call to one of the oad() interface functions displays the attributes and relationships of the managed Object, modNet. (The actual line numbers and variable names might be different in the current version or test case being used.)

```
(gdb) b 111
Breakpoint 2 at 0x8063e39: file testcase.cpp, line 111.
(gdb) c
Continuing.
Breakpoint 2, main (argc=1, argv=0xbfff9f74) at testcase.cpp:123
111  oaBusNet *busNet1 = oaBusNet::create(block,
    oasScalarName(ns,"busNetA1"),3,20,2);
(gdb) call oad(modNet)
<oaModScalarNet isPreferredEquivalent='t' id='b75c0613' name='N2'
    sigType='signal'>
    <mod ref='#b75c0192' name='Sample'/>
    <topMod ref='#b75c0192' name='Sample'/>
    <instTerms>
        <oaModInstTerm ref='#b75c0693' termName='b'/>
    </instTerms>
    <occNets>
        <oaOccScalarNet ref='#b75c0713' name='N2'/>
    </occNets>
    <oaModScalarTerm id='b75c0793' name='T2' isInterface='t'
        position='UINT_MAX' termType='input'>
        <mod ref='#b75c0192' name='Sample'/>
        <topMod ref='#b75c0192' name='Sample'/>
        <occTerms>
            <oaOccScalarTerm ref='#b75c0813' name='T2'/>
        </occTerms>
    </oaModScalarTerm>
</oaModScalarNet>
```

4.1.3 Managed Object address

The value of a managed Object variable, i.e., its handle in hex, can be passed as an argument to oad() – or the non-overloaded synonym, oada() – will also produce a traversal and display of that Object. This is useful because handles to other Objects frequently appear in Object associations. Those hex values may not correspond to any active code variables; yet they can still be traversed.

For example, the instTerms relationship in the modNet display above (4.1.1) shows a reference to a ModInstTerm handle. The following oad call uses that Object address directly to display that ModInstTerm:

```
(gdb) call oad(0xb75c0693)
<oaModInstTerm id='b75c0693' numBits='1'>
    <mod ref='#b75c0192' name='Sample'/>
    <topMod ref='#b75c0192' name='Sample'/>
```
4.1.4 Utility object

Passing the name of a code variable that represents an OA utility object to `oad()` will display just that utility object.

This gdb session fragment lists a few lines of code preceding a breakpoint, then calls for oad display of a few of the utility objects from that code:

```
(gdb) list 87
82  oaParam paramF("paramF", 2.435);
83  pa.append(paramI);
84  pa.append(paramF);
85  pa.append(paramI);
86  oaTransform xf(34, 65, oacMX);
88  oaBusNet *busNet1 = oaBusNet::create(block,
oaScalarName(ns,"busNetA1"), 3,20,2);
90  oaBusTerm *busTerm1 = oaBusTerm::create(busNet1,
oaScalarName(ns,"busTermA1"), 4,21,2);
(gdb) call oad(xf)
<oaTransform x='34' y='65' orient='MX'/>
(gdb) call oad(paramF)
<oaParam name='paramF' type='doubleParam' val='2.435000'/>
(gdb) call oad(pa)
<ParamArray size='3' numElements='7'>
<oaParam name='paramInt' type='intParam' val='2435'/>
<oaParam name='paramF' type='doubleParam' val='2.435000'/>
<oaParam name='paramInt' type='intParam' val='2435'/>
</ParamArray>
```

The non-overloaded `oadu()` can be used when the debugger cannot pick the right overload. This function has the user provide the `address` of the variable plus an explicit `oaType` name. The `oadu()` interface is useful, for example, with

- the .NET debugger, which generally cannot identify the overloaded symbols in a call
- `gdb()`, which cannot find an overloaded template symbols (such as Complex and LookupTbl)

The following session fragment illustrates this problem:

```
(gdb) call oad(complexF)
Cannot resolve function oad to any overloaded instance
(gdb) call oadu(&complexF, "oaComplex<oaFloat>")
<?xml version='1.0' encoding='utf-8'?>
<si2oadebug>
<oaComplex oaComplex='(5.6,7.8)'/>
</si2oadebug>
```
If the exact name of the `oaType` cannot be recalled, the debugger can be used to provide that information about the code variable. Note in the following example, the namespace qualifier is reported by `gdb` but deleted from the Type name for the purpose of calling `oad()`:

```plaintext
(gdb) call oad(vd2DTbl)
Cannot resolve function oad to any overloaded instance
(gdb) whatis vd2DTbl
type =
   OpenAccess_4::oa2DLookupTbl<int,int,OpenAccess_4::oaViaDefArrayValue*>  
(gdb) call oadu( &vd2DTbl, "oa2DLookupTbl<int,int,oaViaDefArrayValue*>" )
...
```

## 4.2 Command Functions

An `oad()` overload – or the non-overloaded synonym, `oado()` – can also be passed two string arguments representing a "command" category and the syntax of that command. Such commands produce an action or change the current options state of `oad` in some manner.

### 4.2.1 Option

Calling `oad("o","optionname=value")` will set the option with the indicated name to the value following the `=` character (5).

If the second arg specifies an invalid option name or the empty string, a list of options will be displayed, along with a prompt for a valid name and value.

### 4.2.2 Read

Calling `oad("r","filepath")` will read the indicated relative or absolute file path and overwrite the current option settings with those found in that file.

If the empty string, the user will be prompted for the filename. The format of the file must be compatible with that produced by the "write" command; however, only those options to be changed need to be included.

If the second argument is the empty string, a prompt will be made to the command line for its value.

### 4.2.3 Write

Calling `oad( "w", "filepath" )` will write the current options settings to the indicated path.

This will create a usable template in the correct format for option settings files, such as the `.si2oadoptions` file of initialization settings read automatically by `oad` (5.1). In situations where several options must be changed, instead of using the "o" command to change them individually, it can be more convenient to:

- write an options file with the "w" command,
- edit it in a separate window,
• and then read in the changes with the "r" command.

If the second argument is the empty string, a prompt will be made to the command line for its value.

4.2.4 Help

Calling `oad("h","")` will display the current version information along with a terse summary of the oad interface calls available. The arg is ignored.

4.2.5 Regex

Calling `oad("r","pathExpression")` will set up regex matching for the next call to display model data (or all subsequent calls if `regexsticky==t`). The second arg is a `pathExpression` (6).
5 Options

A number of options are available to control

- locations for components used by the tool both for initial setup and while it is running
- various aspects of the display, including
  - the output target (console or browser)
  - communications options
  - output filters to attenuate the output generated (which can get voluminous on large designs, causing large XML or HTML files that take a browser some minutes to load and process)

Table 7 lists the various options available. Subsequent sections explain additional semantics and dependencies, as well as different ways to set options during the course of program execution.

5.1 Defaults

All options are initialized with a set of default settings in the Si2OADebug.

When the first call to an oad() interface function is made, an attempt is made to read a file of options settings.

- The first location checked is, ./.si2oadoptions
- If no such file exists, the next location tried is, $HOME/.si2oadoptions.
- If neither file exists, the default option value settings (Table 7) hard-coded in the program are used.

For example,

(gdb) call oad()
[si2oad] Initial oad load: Attempting to read options from .si2oadoptions
[si2oad] ...Attempting to read options from /home/d/.si2oadoptions
[si2oad] ...No options file found. Using default options.

5.2 Changing Options

At any time during execution of the application, option settings can be changed.

5.2.1 Setting Individual Options

Individual options may be changed interactively using, oad("o", "optionname=value"). If the second argument is the empty string (or NULL, for debuggers that can use the overloaded interface) then the user will be prompted interactively for the name and value. The following fragment from a debugger session illustrates changing individual option settings from the console.

28 si2oadUserGuide.odt
(gdb) b 124
Breakpoint 2 at 0x8063b8d: file testcase.cpp, line 124.
(gdb) c
Continuing.
Breakpoint 2, main (argc=4, argv=0xbfffd184) at testcase.cpp:124
124 cout << "got ModNet" << modNet << endl;
(gdb) call oad(modNet)

<oaModScalarNet isPreferredEquivalent='t' id='b75d0612' name='N1' numBits='1' sigType='signal'>
    <instTerms>
        <oaModInstTerm ref='#b75d0692' termName='a'/>
    </instTerms>
    <occNets>
        <oaOccScalarNet ref='#b75d0712' name='N1'/>
    </occNets>
</oaModScalarTerm> # sigType included even though the default value

<oaModScalarNet isPreferredEquivalent='t' id='b75d0792' name='T1' isInterface='t' position='UINT_MAX' numBits='1' termType='input'>
    <net ref='#b75d0612' name='N1'/>
</oaModScalarTerm> # termType included even though the default value

(gdb) call oado("0", ")

OptionName       CurrentValue       Description
---------------   ----------------   ------------------
  output           gui             # Output sent to [gui|console].
  indent           1               # Indent increment representing ownership
  url              http://localhost # URL of the nanowww domain
  port             0               # Port for nanoserver on localhost [0 = OS choice]
  docroot          ../si2dis       # wwwserver root location
  browser          firefox         # Browser (path if not in your $PATH)
  outputpath       ~/si2oad.7425   # Path location for generated output .xml & .html
  deleteoutputpath t             # Delete outputpath when GUI returns to console
  .xml             si2oad.xml     # XML output filename
  .xsl             si2oad.xsl     # Path to the XSL source to transform raw XML into HTML
  .js              si2oadheader.js # Path to the Javascript header file (si2oadheader.js)
  apilink          apilinkTable.js # Path to the apilinktable file (apilinktable.js)
  .css             si2oad.css     # Path to the CSS stylesheet for the HTML
  oadoc            /home/226p052/doc/oa/html # Path or URL to OA doc/oa/html root
  xslexe           $HOME/.../XML/bin/Xalain # Path to the XSLT processor executable
  xsllib           $HOME/.../XML/lib # Colon-separated paths to libs needed by XSLT
  html             si2oad.html # Filename of HTML transform of XML dump
  namespace        Native         # oaNamespace to use for rendering all mapped names
  verbose          2              # Verbose printing of OADebug progress/status messages
  dllobserver      0              # Path to dll with Observers implementation (0 if none)
  maxextdata       16             # Max bytes of Prop/AppDef to display in window (0 if no limit)
  sysdatadisp      0              # Path to exe to display Prop/AppDef data (0 if none)
  dilldatadisp     0              # Path to dll to display Prop/AppDef data (0 if none)
  sysmcmapdisp     0              # Path to exe to display CMap data (0 if none)
  dillcmapdisp     0              # Path to dll to display CMap data (0 if none)
  dofalsebools     t              # Do Boolean attributes if false
  dodefaults       f              # Do attributes with default values
  dosinglebits     t              # Do SingleBitMembers for BitNets
  doredundants     t              # Do 'redundant' attributes that can be inferred
  doblock          t              # Do the Block Domain when dumping a Design
  domod           t                # Do the Module Domain when dumping a Design
  doocc           t                # Do the Occurrence Domain when dumping a Design
  expandocc       f                # Expand the Occurrence tree for each OccInst
doparasitics t # Do parasitics
loadnetworks f # Load all ParasiticNetworks
donullobjs t # Do NULL Object references
dobbox t # Do BBox report (and calculation side-effects)
casesensitive t # Default case-sensitivite setting for regex
regexstickey t # If t & regex set, it applies to all subsequent oad()
xdelimescape \ # Regex escape char used with itself and type, name delims
dxelimtype b # Regex delimiter char between Objects [b means blank]
xelimname = # Regex delimiter char between name components
delimLVC = # Delimiter char between L/C/V name components
maxownerlevels 0 # Max owner chain depth to display (0 shows all data)
maxcollentries 0 # Max # of Collection entries to display at a time
dllsorters
../si2dis/si2oadSortName,../si2dis/si2oadSortOaTypeName,../si2dis/si2oadSortInfoType

Use t or f for Booleans. Use first 3 letters for namespace:
Native Cdba Verilog Vhdl Spice Spf Spef Lef DefUnix Windoze

TYPE optionName newValue<ENTER> [/ to CANCEL] dodefaults f

[gdb] call oado("w", "save.opts")
Save current option settings

(gdb) shell sed 's/^output *.*/output gui/' save.opts >gui.opts
Edit one of the options in that file

[gdb] call oado("r", "gui.opts")
Read file to reset options

5.2.2 Options File

Multiple options may be changed interactively using, oad("r","filename"). The file name specified
must contain option settings in the same format as that created by, oad("w","filename"). Any
number of option settings may be included in the file. These overwrite existing option settings so it is
not necessary to have the complete set of values in the file. Option assignments in a file are read and set
sequentially; so if the same option value is set more than once in a file, the option will end up with the
last value set. The following gdb session fragment first writes all option settings to a file, then uses sed
to change one option, then reads that file to effect the option change.

(gdb) call oado("w", "save.opts")  # Save current option settings
(gdb) shell sed 's/^output *.*/output gui/' save.opts >gui.opts
(gdb) call oado("r", "gui.opts")  # Read file to reset options
<table>
<thead>
<tr>
<th>Option ID</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Verbose</strong></td>
<td>**INT: 0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Output Target:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>output</strong></td>
<td>**CHOICE: gui</td>
<td>console</td>
</tr>
<tr>
<td><strong>Output Rendition Controls</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>namespace</strong></td>
<td><strong>NAMESPACE:</strong> Native</td>
<td>Verilog</td>
</tr>
<tr>
<td></td>
<td><strong>DEFAULT = Native</strong></td>
<td></td>
</tr>
<tr>
<td><strong>indent</strong></td>
<td><strong>INT</strong></td>
<td><strong>DEFAULT = 1</strong></td>
</tr>
<tr>
<td><strong>alwayssort</strong></td>
<td>**BOOL: t</td>
<td>f</td>
</tr>
<tr>
<td><strong>delimLCV</strong></td>
<td><strong>CHAR</strong></td>
<td><strong>DEFAULT = /</strong></td>
</tr>
</tbody>
</table>

**If (output == gui)**

| | | |
| **url** | **URL ** | **DEFAULT = http://localhost** | URL for nanowww communications |
| **port** | **INT** | **DEFAULT = 0** | Port # for the nanowww server host |
| **docroot** | **PATH** | **DEFAULT = ..\si2dis** | Document root for the nanoserver |
| **outputpath** | **PATH** | **DEFAULT = ..\si2dis** | Directory for XML & HTML outputs |
| **xml** | **FILENAME** | **DEFAULT = si2oad.xml** | Output XML filename |
| **browser** | **PATH** | **DEFAULT = mozilla** | Browser executable (for GUI output). |
| **xsltype** | **CHOICE: none | browser | processor** | **DEFAULT = processor** | XSL transformation type |
### If \((output==\text{gui}) \land (\text{xsltype}==\text{browser|processor})\)

<table>
<thead>
<tr>
<th>.xsl</th>
<th><strong>PATH</strong></th>
<th>DEFAULT = si2oad.xsl</th>
<th>Path to .XSLT source to transform raw XML into HTML (ignored if the processor option is set to &quot;none&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.js</td>
<td><strong>PATH</strong></td>
<td>DEFAULT = si2oadheader.js</td>
<td>Path to Javascript header (normally, do not change)</td>
</tr>
<tr>
<td>apilink</td>
<td><strong>PATH</strong></td>
<td>DEFAULT = apiLinkTable.js</td>
<td>Path to Javascript apilinktable (normally, do not change)</td>
</tr>
<tr>
<td>.css</td>
<td><strong>PATH</strong></td>
<td>DEFAULT = si2oad.css</td>
<td>Path to the CSS stylesheet for HTML</td>
</tr>
<tr>
<td>oadoc</td>
<td><strong>PATH</strong></td>
<td>DEFAULT = (\text{<a href="https://openeda.si2.org/si2_online/oa226_p052api/doc/oa/html/index.html%7D%5C">https://openeda.si2.org/si2_online/oa226_p052api/doc/oa/html/index.html}\</a>)</td>
<td>URL to OA API doc tree html/root</td>
</tr>
</tbody>
</table>

### If \((output==\text{gui}) \land (\text{xsltype}==\text{processor})\)

<table>
<thead>
<tr>
<th>xslexe</th>
<th><strong>PATH</strong></th>
<th>DEFAULT = ../..XML/bin/Xalan</th>
<th>Path to the XSLT processor executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslib</td>
<td><strong>PATHLIST</strong></td>
<td>DEFAULT = ../..XML/lib</td>
<td>Colon-separated list of paths to libs needed by the XSLT processor</td>
</tr>
<tr>
<td>.html</td>
<td><strong>PATH</strong></td>
<td>DEFAULT = si2oad.html</td>
<td>Filename (no path) of HTML transform of XML (normally, do not change this)</td>
</tr>
</tbody>
</table>

### Output Data Filters: Parsitics

<table>
<thead>
<tr>
<th>doparasitics</th>
<th><strong>BOOL</strong>: t</th>
<th>f</th>
<th>DEFAULT = t</th>
<th>Display (true) or omit (false) all parasitics</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadnetworks</td>
<td><strong>BOOL</strong>: t</td>
<td>f</td>
<td>DEFAULT = t</td>
<td>Load all ParasiticNetworks automatically</td>
</tr>
</tbody>
</table>

### Output Data Filters:

32 si2oadUserGuide.odt
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dofalsebools</td>
<td>BOOL</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>dodefaults</td>
<td>BOOL</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>dosinglebits</td>
<td>BOOL</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>doredundants</td>
<td>BOOL</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>donullobjs</td>
<td>BOOL</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>dobbox</td>
<td>BOOL</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>maxownerlevels</td>
<td>INT</td>
<td>1</td>
<td>Traverse and display only the specified number of nested levels of the Object ownership hierarchy.</td>
</tr>
<tr>
<td>maxcollentries</td>
<td>INT</td>
<td>1</td>
<td>The maximum number of members of a Collection to display at a time (especially useful with large Collections of Objects)</td>
</tr>
</tbody>
</table>

**Output Filters: Domains**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doblock</td>
<td>BOOL</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>domod</td>
<td>BOOL</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>doocc</td>
<td>BOOL</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>expandocc</td>
<td>BOOL</td>
<td>t</td>
<td>f</td>
</tr>
</tbody>
</table>

**Output Filters: Iters**
<table>
<thead>
<tr>
<th>iterinsts</th>
<th>INT</th>
<th>DEFAULT = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR bits:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0000</td>
<td>oacInstIterAll</td>
<td></td>
</tr>
<tr>
<td>0x0001</td>
<td>oacInstIterSingleBit</td>
<td></td>
</tr>
<tr>
<td>0x0002</td>
<td>oacInstIterNotImplicit</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iterinsterms</th>
<th>INT</th>
<th>DEFAULT = 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR bits:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0000</td>
<td>oacInstTermIterAllNotHidden</td>
<td></td>
</tr>
<tr>
<td>0x0001</td>
<td>oacInstTermIterSingleBit</td>
<td></td>
</tr>
<tr>
<td>0x0002</td>
<td>oacInstTermIterNotImplicit</td>
<td></td>
</tr>
<tr>
<td>0x0004</td>
<td>oacInstTermIterEquivNets</td>
<td></td>
</tr>
<tr>
<td>0x0010</td>
<td>oacInstTermIterAll</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>internets</th>
<th>INT</th>
<th>DEFAULT = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR bits:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0001</td>
<td>oacNetIterAll</td>
<td></td>
</tr>
<tr>
<td>0x0002</td>
<td>oacNetIterSingleBit</td>
<td></td>
</tr>
<tr>
<td>0x0010</td>
<td>oacNetIterPreferred</td>
<td></td>
</tr>
<tr>
<td>0x0020</td>
<td>oacNetIterNotImplicit</td>
<td></td>
</tr>
<tr>
<td>0x0040</td>
<td>oacNetIterGlobal</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iterterms</th>
<th>INT</th>
<th>DEFAULT = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR bits:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0001</td>
<td>oacTermIterAll</td>
<td></td>
</tr>
<tr>
<td>0x0002</td>
<td>oacTermIterSingleBit</td>
<td></td>
</tr>
<tr>
<td>0x0004</td>
<td>oacTermIterBundle</td>
<td></td>
</tr>
<tr>
<td>0x0010</td>
<td>oacTermIterNot Implicit</td>
<td></td>
</tr>
</tbody>
</table>

**Regex Controls:**

<table>
<thead>
<tr>
<th>casesensitive</th>
<th>BOOL: t</th>
<th>f</th>
<th>DEFAULT = t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case-sensitive (true) or case-insensitive (false) matching of each Object name</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>regexsticky</th>
<th>BOOL: t</th>
<th>f</th>
<th>DEFAULT = t</th>
</tr>
</thead>
<tbody>
<tr>
<td>If a regex pathExpression has been set via call to oad(&quot;x&quot;,&quot;...&quot;), then use regex matching:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• (if f) Once only, on the very next oad call to display data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• (if t) For all oad calls (until regexsticky is set to f)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xdelimescape</th>
<th>CHAR</th>
<th>DEFAULT = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Escape character to enable xdelimtype and xdelimname characters to be used as literal characters. 0 means none.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xdelimtype</th>
<th>CHAR</th>
<th>DEFAULT = b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delimiter, at least one of which, that must precede each (except the first) typeRegex in a regex pathExpression. The letter b indicates a blank.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xdelimname</th>
<th>CHAR</th>
<th>DEFAULT = =</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delimiter, exactly one of which, that must precede each nameRegex in a regex pathExpression.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**User Code Extensions:**
<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extdatamax</td>
<td>UINT</td>
<td>DEFAULT = 8</td>
<td>The maximum number of bytes of Prop or AppDef data to display embedded in the XML or HTML output.</td>
</tr>
<tr>
<td>sysdatadisp</td>
<td>PATH</td>
<td>0</td>
<td>emacs</td>
</tr>
<tr>
<td>dlldatadisp</td>
<td>PATH</td>
<td>0</td>
<td>DEFAULT = 0</td>
</tr>
<tr>
<td>syscmapdisp</td>
<td>PATH</td>
<td>0</td>
<td>emacs</td>
</tr>
<tr>
<td>dllcmapdisp</td>
<td>PATH</td>
<td>0</td>
<td>DEFAULT = 0</td>
</tr>
<tr>
<td>observer</td>
<td>PATH</td>
<td>0</td>
<td>DEFAULT = 0</td>
</tr>
<tr>
<td>dllsorters</td>
<td>PATHLIST</td>
<td>0</td>
<td>DEFAULT = 0</td>
</tr>
</tbody>
</table>

### 5.3 Option Details

#### 5.3.1 Shell Meta-Character Expansion

Options that represent path names in the file system all can use Bourne shell meta-characters, which will automatically get expanded to an actual path name by the POSIX `wordexp()` function. Useful metas include those in the table below:

<table>
<thead>
<tr>
<th>Character</th>
<th>Expanded to</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>The current directory</td>
</tr>
<tr>
<td>..</td>
<td>The current parent directory</td>
</tr>
<tr>
<td>$VARIABLE</td>
<td>The current value for an environment variable</td>
</tr>
<tr>
<td>~</td>
<td>Users $HOME director</td>
</tr>
</tbody>
</table>

#### 5.3.2 Setting Options

*Table 9* illustrates how to set options from the command line.
<table>
<thead>
<tr>
<th>Task</th>
<th>Input to gdb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send data display output to console</td>
<td>call oad(&quot;o&quot;,&quot;output=console&quot;)</td>
</tr>
<tr>
<td>Send data display output to browser</td>
<td>call oad(&quot;o&quot;,&quot;output=gui&quot;)</td>
</tr>
<tr>
<td>Omit any false bool attributes from the display</td>
<td>call oad(&quot;o&quot;,&quot;dofalsebools=f&quot;)</td>
</tr>
<tr>
<td>Limit console messages to critical information only</td>
<td>call oad(&quot;o&quot;,&quot;verbose=0&quot;)</td>
</tr>
<tr>
<td>Print as much status/state information as possible</td>
<td>call oad(&quot;o&quot;,&quot;verbose=9999&quot;)</td>
</tr>
<tr>
<td>Set the default for regex-name matching to case-insensitive</td>
<td>call oad(&quot;o&quot;,&quot;casesensitive=f&quot;)</td>
</tr>
<tr>
<td>Omit the Module Domain from the display</td>
<td>call oad(&quot;o&quot;,&quot;domod=f&quot;)</td>
</tr>
</tbody>
</table>

### 5.3.3 verbose

The `verbose` option may be set to any integer $\geq 0$ to control how much diagnostic output is generated. As the value increases, more messages are printed to the `console` (regardless of the setting of the `output` option). A setting of 0 prints only important errors to that must be seen.

### 5.3.4 output

This option determines the target of the output generated by calls to `oad()`. If the value is,

- `console`, the XML output will be sent to `stdout`.
- `gui`, the output will be sent to a browser for display as either XML or HTML, depending on the setting of the `xsltype` option. Once the browser has been launched and data is being displayed in GUI mode, many additional requests for additional RTM data can be made of the nanoserver driving the display. At some point, however, the user will want to return control to the console (i.e., `stdin`) in order to communicate with the debugger again. To switch back to the console from the GUI,
  - Press the **RETURN CONTROL TO CONSOLE** button at the top
  - If all browser instances were deleted before this button could be pushed, simply open a new browser to URL: `http://localhost:PPPPP/quit` (that is, with same URL and Port options that are listed in the console output just prior to opening of the original browser instance).

Switching back with the button or a “quit” URL request to the nanoserver will gracefully return control to the debugger at the exact point where the `oad()` call was executed without the need to send an interrupt signal.

### 5.3.5 indent

If the option `output=console`, the indent option sets the number of spaces to indent each successive level of nested containment or ownership.

In the case `output=gui`, the file named in the `.css` option (5.1.5) controls the size of indents in the HTML displayed by the browser.
5.3.6 alwayssort

If the value of this alwayssort=t then Collection entries will be sorted by default using the first sorter DLL in the list. If no sorterDLLs have been loaded successfully, this option is set to f automatically. Using this option is the only way to coerce the stand-alone si2dis program to sort Collections.

5.3.7 url

When GUI is selected for the display output, a skeletal HTTP server (nanoserver) is started and becomes the source of data for the interaction with the browser. All data is communicated to the browser from this nanoserver. By default, localhost is used as the URL along with the port number specified by the port option.

During the life of a display, the user may make requests for additional Objects that were not included in the original set of data being displayed. For example, ifoad(someNet) invokes the browser to display that Net, and the user clicks on the block=0xbc7784 association reference to that Net's Block, Javascript on the page will cause a request to be sent to Si2OADebug for the data using the same URL specified by the url option.

5.3.8 port

The port used by the nanowebserver can be explicitly assigned using this option. Typically, however, a 0 is used to let the operating system pick an unused socket to avoid interfering with other socket traffic.

5.3.9 docroot

If GUI mode is selected (output=g), and XSL processing will be used to convert the XML output to HTML, then several files are required for the browser to display that HTML properly. The code will check a manifest of the files needed each time the docroot is set — including the first time on start-up, the default “./si2dis” default location in case the .si2oadoptions file does not change that.

The current manifest is defined by the docrootFiles array in the code; for example, for 1.3.2 it was:

```c
const char *docrootFiles[] = {
    "si2oadUserGuide.pdf",
    "si2logotrim.gif",
    "si2oad.xsl",
    "si2oad.css",
    "apiLinkTable.js",
    "si2oadheader.js",
    "favicon.ico",
    "regexChoices.html",
    "quit.html"
}
```

These files can be located in any directory in which the owner of the process executing Si2OADebug has read/execute permission (since the browser must be able to change to that directory and read the files).
The code will not copy the files there automatically, but it will note each file that is missing from the manifest in a console error message.

Typically, the same docroot will be defined for any number of users simultaneously running OAD since the manifest files do not change for a given version of the code. The outputpath (5.1.2), on the other hand, almost always needs to be a different directory, unless only one user at a time will be running the tool.

5.3.10 outputpath

When output=g, this directory specifies the location of the output data (including error messages that display in pop-up windows) generated by each oad() call, as well as calls within the GUI environment that cause incremental data to be sent by the nanoserver to the browser. If more than one user could be running OAD at the same time, a unique outputpath must be created for each user, since the same output file names are used. Even if each user promises to use a different .xml and .html names, extension and Cmap data, as well as error message content, all use default, currently changeable names. Since multiple users might well be using a common docroot (5.1.1), the outputpath should be different from that directory. A popular way to uniquify this directory name is to attach the special token, $PID, to the name. For this option only, the code will replace this special text token with the current process ID. This replacement happens before the string is then expanded for shell meta characters (5.1.1).

5.3.11 browser

The value of this option sets the executable to be used for GUI output. If that executable is not in $PATH then a path must be specified that is either absolute or relative to the location of Si2OADebug's library. Raw XML display and (browser client-side) XLST processing has been successfully tested with:

- Firefox 1.0.7, 1.5.0.1 (Linux)
- Mozilla 1.4 (Linux)
- Netscape 7.2 (Win2K)
- IE 6.0 (Win2K)

5.3.12 xsltype

Meaningful only if output=gui, the following xsltype settings are recognized:

- none: The output will be raw XML.
- browser: The native XSLT engine of the browser will be used.
- processor: An external XSLT processor (the path set in the xslexe option) will be used.

5.3.13 .css

The path to the CSS style file that governs the rendition of the HTML in the browser in the case that option output=gui.

5.3.14 xslexe

If xsltype=processor, the value of this option is the name of the executable that will be used to transform the output XML into HTML for use by the tool named in the browser option. If this XSLT
processor is not in the PATH, then a path name must be specified that is either absolute or relative to the location from which the Si2OADebug library was loaded.

5.3.15 xlslib

If xsltype=processor, the value of this option is the colon-separated path list to be appended to the LD_LIBRARY_PATH to resolve shared libraries requirements (if any) of the xslexe executable. Each path in the list must be either absolute or relative to the location from which the Si2OADebug library was loaded.

5.3.16 loadnetworks

Setting loadnetworks=t wil force all ParasitcNetworks (and their partitions, if any) to be loaded by oad automatically.

Even if this option is f, individual Networks can be loaded interactively from the GUI by clicking on the LOAD hot link. Such Networks are automatically unloaded immediately after the display so the runtime state remains as it was prior to the oad() call.

5.3.17 dobbox

Displaying a bbox causes a recalculation, which can change the RTM state. This violates the “do no harm” precept that is a core of oad. However, if the user needs to see this data updated dynamically in the output, setting dobbox=t will cause all bbox data to be calculated. Even in this case, however, bboxes will be updated only to the extent that the OA API can manage. For example, requesting the bbox of a Block will not cause binding of all its Inst to unbound masters; hence, even a recalculated bbox might not be accurate if changes were made to an unbound Inst's master since the data was last stored on the Inst.

5.3.18 maxownerlevels

This option is perhaps the most valuable to maintain fast response using a GUI to traverse large Designs. If maxownerlevels=0, the Object ownership hierarchy will be traversed completely, sending all data to the output stream. For large Designs, this can be a huge amount of XML.

If the output target is a browser, the XSLT process produces an even larger amount of HTML, which will result in very sluggish browser response to the point of crashing some browsers with hundreds of megabytes of HTML. This problem can be controlled by setting maxownerlevels to some low number, like 1. This results in very fast loading for even the largest Designs.

Limiting the depth of traversal when the output is being sent to the console can also be valuable, enabling incremental selection of a more limited set of interesting Objects for extended traversal. Otherwise, the display terminal can become quickly overwhelmed with data, leaving only the last few bits of data. The complete output can be captured even on the console if

- the console tool's history buffer has been set to “infinite”.
- an emacs shell buffer is used.
On the other hand, if the goal is to capture or archive a complete design in XML, sending the output to a file, then it may be necessary to set \texttt{maxownerlevels=0}, or to a number larger than 1. Fortunately, capturing just the XML takes only a fraction of the time it takes to transform it to HTML and load that into a browser.

When this option is set to limit depth of traversal, when that limit is reached, instead of data as the indented contents of a parent Object, a special sentinel tag is planted to indicate the model extends deeper at that point. When \texttt{output=gui}, the sentinel itself will not display; however, clicking on the leading + for the Object entry that contains such a sentinel will cause a real-time callback to the nanoserver for automatic traversal and display of an additional \texttt{maxownerlevels} number of container levels. This enables incremental display of only the parts of the RTM of interest to the developer, thereby reducing the time it takes for any given XSL transformation and browser load of HTML data.

5.3.19 \texttt{expandocc}

To avoid altering the RTM, oad will normally not call \texttt{getMasterOccurrence()} for any Insts or \texttt{OccModuleInstHeaders}. If \texttt{expandocc=t}, then for each \texttt{oaOccInst}, an Occurrence hierarchy will automatically be bound into the RTM with the owner \texttt{OccInst} or \texttt{OccModuleInstHeader} as the container for the resulting Occurrence. Any \texttt{OccInsts} and \texttt{OccModuleInstHeaders} within that Occurrence will also be expanded, recursively to the frontier.

Even if this option is \texttt{f}, individual \texttt{OccInst} master Occurrences can be expanded interactively by clicking on the \texttt{EXPAND} hot links. Occurrence trees expanded in this fashion will be destroyed automatically when the current GUI session ends, thereby leaving the RTM in its original, unexpanded state.

If the RTM already had an Occurrence master expanded (by the application's activities) at the time oad is invoked, that Occurrence tree will be displayed regardless of the \texttt{expandocc} setting, since it already existed in the data.

5.3.20 \texttt{dodefaults}

The default values of many attributes can be inferred or looked up easily. Setting the \texttt{dodefaults} to \texttt{f} can further reduce the volume of output. Examples of such attributes are,

- \texttt{isPreferredEquivalent} (defaults to true, unlike most \texttt{oaBooleans})
- \texttt{double, float, int, TimeStamp, if} == 0
  (not overridden if 0 is not the default value)
- \texttt{Strings} if == ""
- \texttt{RefCount}s if == 1
- Most enums if == 0 (ie, the first enum in the list); but overridden when that is not a reasonable default value
- \texttt{OccNet/OccInst pathName} if the same as the \texttt{name}
- \texttt{Arrays} if both \texttt{size} and \texttt{numElements} == zero
5.3.21 doredundants

The values of severay attributes (typically those common to a main category of Objects, such as DesignObjects or DMObjects) can be inferred from their context in the display. Setting the doredundants option to \( \ell \) suppresses these from the display. This can appreciably reduce the volume of output providing for faster output generation as well as making it easier to pick out the more important attributes from a sea of data. Examples of such attributes include,

- References to the owner/container object if it is displayed above
- The domain of an Object
- isContainer for DM Container Objects
- isPreferredEquivalent if no equivalent nets
- The count/numObjects/numBits in a Collection since the list itself appears in the display
- "has*" Booleans, since the Object itself will appear in the display
- OccNet "span" of only the OccNet itself

5.3.22 Extension Data

Setting the option, \texttt{maxextdata}, to a reasonably small number of bytes can prevent cluttering the output with potentially voluminous extension data values of AppProp, DataAppDef, and VarDataAppDef extension Objects. If that value for such an extension exceeds the set limit, a \ldots will be included after the bytes displayed to indicate there is more data in the value. Clicking on this \ldots will invoke a confirmation dialog. The complete data value will be displayed in a separate window if the value for option

- \texttt{sysdatadisp} is set to the name of an executable to be invoked (in the background with &) via system() call in a separate process. The data is saved to a temporary file, which is passed to the executable as its first argument.
- \texttt{dlldatadisp} is set to the name of a shared object (DLL) to be loaded into the current process space. The DLL must use C linkage to declare the entry point, ""

In either case, an absolute path to the data display tool must be used if that tool is not in, respectively, the PATH or shared object load path for the process.

5.3.23 CMap Data

Setting the option, \texttt{maxextdata}, to a reasonably small number of bytes can prevent cluttering the output with potentially voluminous CMap data. If that data exceeds the set limit, a \ldots will be displayed.

1. Edit the .si2oadoptions file. Set the \texttt{sysdatacmap} option to \texttt{emacs} (or your favorite text editor) using an absolute location if not in the PATH of the current process.

2. Run the testcase, test1.cpp, setting a breakpoint just before the deleteSupplyDemand() is executed for the CMap.

```
make oad TEST=test1
b 710
```
3 In the GUI open the first design junk/schematic

- open the Block
- open the CMAP
- slide to right of display line and click on ...
- ask for layer 201

This should open the data in emacs as a separate system() call.

Or you can edit .si2oadoptions and set dllcmapdisp to the dll for it that I included (which is just a system call to emacs anyway): si2oadCMapViewer.so
6 Regular Expressions

In addition to the various attribute filter options, regular expressions can be used to limit output to a subset of matched Objects (and their surrounding context). Considerable familiarity with OpenAccess “owner” relationships is necessary to use Si2OADebug regular expression filters effectively. The OA Information Model Schema Diagrams and Si2 Course UML models are useful references to aid visualization of these relationships.

6.1 Summary

To control matches for output filtering, the oad() function is called with the “x” command along with a single pathExpression composed of one or more componentExpressions, which are in turn composed a typeRegex and an optional nameRegex in the following form:

\[
\text{typeRegex} [=\text{nameRegex}] \ [\text{typeRegex} [=\text{nameRegex}] \ldots]
\]

As shown above, a componentExpression consists of

- one typeRegex designed to match the name of an OA Object Type

  These Type names are the names of OA classes, but without the "oa" prefix. Every typeRegex is matched as case-insensitive. Consequently, the pathExpression "net" and "Net" both match all oa*Net* Types, which includes, for example, oaScalarNet as well as oaBusNetDef.

- an optional nameRegex that is designed to match names of Objects of the Type(s) matched by the typeRegex

  With no nameRegex, all names for the Type are matched (equivalent to .*). For Objects with no name, such as Blocks, a nameRegex should not be entered. Case-sensitivity of the nameRegex matching is set by the casesensitive option in the options file (5.2.2), and may be overridden by a prefix to the pathExpression (6.3.3).

6.2 Syntax Details

After a pathExpression is specified (set with the oad() “x” command), the next oad() call to display data will use regex matching:

```plaintext
call oad( "x", "pathExpression" )
call oad(rootObject)
```

Regex matching is enabled and disabled independently of defining and changing the pathExpression itself. The regexsticky option (6.3.2) can be used to force all subsequent oad() calls to use regex matching; otherwise, only the call immediately following the oad("x","...") command that set the expression will use it for matching.

POSIX "extended" (a.k.a. "modern") regex syntax is used for all regular expressions.

6.2.1 Delimiters

Delimiters are used to separate each typeRegex from the preceding one, and each nameRegex, from its typeRegex. These delimiters are set independently using the following options (6.3):

- xdelimtype: 2 or more must precede each successive Object typeRegex (The default is a blank.)
- xdelimname: 1 must separate a typeRegex from its nameRegex. (The default is an equal sign.)

These delimiters can be changed independently, as needed, for any pathExpression.

- If a nameRegex must use two consecutive blanks (the default xdelimtype) to match desired Objects in the data, the xdelimtype must be changed to a different character, one that will not be needed for the regex match in question. The delimiters can be changed as often as necessary in a session, though such changes must be made prior to executing the oad("x","...") command that needs them. Of course, the xdelimtype may not be a letter or number that might appear in an oaType name.

- Similarly, if a nameRegex must use an = because it occurs in the name of Objects to be matched, the xdelimname must be changed, say, to a + (assuming that will not need to be used in the nameRegex).

For example, to match Terms in Nets with consecutive 'B' characters embedded in their names, the following pathExpression could be used:

Consider the following “x” command that will match Nets named as “myNetBB<digit>” and then Terms whose names end in “Erouted”.

```plaintext
call oad( "x", "net$=BB  term$" )
```

However, suppose instead of “BB” in the Net names, consecutive blanks needed to be matched. These two blanks could not be distinguished from the double blank delimiting the end of the nameRegex from the following typeRegex. So a solution would be,

```plaintext
call oad( "o", "xdelimtype=~" )
call oad( "x", "net$=  ~~term$" )
```

If the Net names to be matched instead had an equal sign, it would conflict with the equal sign separating the end of the typeRegex from the start of the nameRegex. So the xdelimname could be changed as follows (note that the xdelimtype could be set back to a blank, if that were no longer to be part of the nameRegex):

```plaintext
call oad( "o", "xdelimtype= " )
call oad( "o", "xdelimname=`" )
call oad( "x", "net$`=  term$" )
```

6.2.2 Owner Hierarchy

When regex matching is in effect, a call to oad(rootObject) begins matching the first componentExpression against the Objects owned by the rootObject in the call (or from the top of the RTM if oad() is invoked without a rootObject). Consequently, the start and order of componentExpressions is critical. If the first typeRegex is not aligned with the start of the ownership
hierarchy, nothing will be matched. For example, the following \textit{pathExpression} will match Terms (of any name), owned by Nets that are named with a single, capital letter – but only from a call starting with a Block, which is the owner of Nets:

\begin{verbatim}
call oad( "x", "net=[A-Z] term=.*" )
call oad(block6);
\end{verbatim}

If any \texttt{rootObject} other than a Block is used, nothing will match. The caller must know the ownership hierarchy of the OA information model to use regex matching effectively. This expression style enables exact targeting of Objects that can appear on Object Types at different levels of the owner hierarchy. For example, IntProps in a Block with specific Types of owners can be selected as follows:

\begin{itemize}
\item Net owners: \texttt{oad( "x", "net intprop" )}
\item ScalarTerm owners: \texttt{oad( "x", "net scalarterm intprop" )}
\item all Objects \textit{directly} owned by Nets: \texttt{oad( "x", "net .* intprop" )}
\end{itemize}

The following \textit{pathExpression} must be used from a call to \texttt{oad()} without a root Object argument since it begins matching the first contained Lib object:

\begin{verbatim}
call oad( "x", "lib Design block .*")
\end{verbatim}

The above expression will match,

\begin{itemize}
\item only Libs (e.g., not the Session) at the top of the RTM
\item only Designs in each Lib (not, for example, Cells, Views, etc.)
\item only the Block in the each Design (but not the Modules or Occurrence)
\item every Object Type (directly) owned by the Block
\end{itemize}

6.2.3 Limitations

Since each successive level of the ownership hierarchy must be matched by each successive component in the \textit{pathExpression}, there is no way to select, for example, "all IntProps, regardless of owner Object, throughout the Block's ownership hierarchy". The IntProps on all Block Nets or those on Terms can be selected but not both at the same time (since Terms and Nets have different owner Objects). Most OA Types have only a single Object Type that can be an owner; so this limitation is only relevant for Objects that can have different owner Types, such as Props, AppDefs (i.e., the instantiations of AppDefs attached to an \texttt{oaObject}), Constraints, Values etc.

If \texttt{output=gui} and display is being controlled incrementally (with \texttt{maxownerlevels} set to some low value), the callback code will attempt to guess where in the owner containment chain the request for more data is taking place. This is done by running \texttt{regexec()} on the request Object's Type, against the \texttt{typeRegex} in each of the \textit{componentExpressions} starting from the beginning until a match is made. While in the large majority of use cases, the match will put reset the the current containment position to the correct level, the guess could be wrong for two reasons:

\begin{itemize}
\item As described above for regex in general, some Objects can have different owners at different levels of the owner hierarchy.
\item A sufficiently general typeRegex could end up matching in an unexpected part of the owner chain.
\end{itemize}
6.2.4 Special Character Semantics

6.2.4.1 Name Semantics

The Lib, Cell, and View names of Designs are concatenated for matching and display purposes using the xdelimname (option) character as a separator. These concatenated names can then be matched using a nameRegex as though this composite were a single name attribute of the Design. For example, if xdelimname is the '=' character, the following calls will match all the Designs in lib6 with AND in the Cell name:

```c
    call oad( "x", " Design=^AND.*=.*=.*" )
    call oad(lib6)
```

The pathExpression, "lib Design=^[lL]ib=.*j.*=schematic$", will match only Designs having a
- Lib name of "Lib" or "lib"
- View name with a "j" in it
- Cell name "schematic"

Bit semantics are treated using the special characters of the current NameSpace in effect (via the namespace option). For example, in a NameSpace where [] are the vector semantics delimiters, the pathExpression above would match any BusNet in any of the three Domains (Block, Module, Occurrence) with
- a base name ending in a capital letter and one digit
- any start value
- stop value of 20
- any single-digit, odd step value.

The $ at the end of the typeRegex above prevents matching Object Type names with characters after the final "t" (such as BusNetDef and BusNetBit). Adding a ^ to the front of the typeRegex would further prevent matching ModBusNets and OccBusNets by restricting matches to Type names starting with "b". In the si2distest/test1.cpp testcase this expression will match only busNetA2[3:20:5] when called using the "block" variable (in the Design named, Lib/Sample/schematic).

6.3 Regex Options

Several option (4.1.4) settings affect regex behavior.
6.3.1 xdelimname

As noted in the “Name Semantics” examples (above), successful matching of LCV names requires use of the delimiter character set by the xdelimname option.

6.3.2 regexsticky

This regexsticky option affects the way oad calls that do not supply a regex are handled. If set to f, those calls will not use regex matching (without deleting any pathExpression), if set to t, then if a pathExpression has been set, it is used in regex matching until a different one is set, or regexsticky is set to f. The call below turns on "sticky":

```plaintext
  call oad("o","regexsticky=t")
```

6.3.3 casesensitive

Case-insensitive matching (for nameRegex only) can be enabled by setting:

```plaintext
(gdb) call oad( "o", "casesensitive=f" )
```

With the above setting, all subsequent calls to oad that use regex matching will ignore case. Regardless of the setting of the casesensitive option, a case override flag may be prepended to the front of a pathExpression to override the casesensitive option setting read from the .si2oadoptions file (or the default, if none) using a syntax of the form,

```plaintext
[{{+|-}{i|s}}  type=[nameRegex] [type=[nameRegex] ...]
```

For example, the following call forces case-insensitive matching:

```plaintext
(gdb) call oad( "x", "+i ^lib$ design$ block net$" )
```

This prefix

* may be separated from the first typeRegex by 0 or more xdelimtype characters
* must start with + or - where,

  ♦ +s and -i both meaning case-sensitive
  ♦ -s and +i both meaning case-insensitive

Note that typeRegex matching against Object Type names is always case-insensitive.

6.3.4 Help Information

Setting the option verbose=2 (or higher) will produce, at the time a pathExpression is defined, details about which objects will be matched and the case-sensitivity status. For example,

```plaintext
(gdb) call oad( "o", "verbose=2" )
(gdb) call oad( "x", "busnet=busNet[A-Z][0-9]\/[.*:20:\[13579\]\]" )
[si2oad] Processing regex
[si2oad] ...No case override since no leading + or -
```
[si2oad] ...Deleted all prior regex entries
[si2oad] ...busnet matches:  BusNet  BusNetBit  BusNetDef  OccBusNet
          OccBusNetBit  OccBusNetDef  ModBusNet  ModBusNetBit  ModBusNetDef

The same verbose level will also produce at run-time a log of which Objects in the RTM are selected and what names are matched, as shown in the following console excerpt (with several lines omitted and replaced by "..."):  

[si2oad] ...MATCH automatically root Object in oad call
[si2oad]    Using regex but at root Object. Skipping CheckNamePattern
[si2oad] ...SKIP TYPE=AppIntDef: NO MATCH to type-regex[0]=busnet (osd=1)
[si2oad] ...SKIP TYPE=IntProp: NO MATCH to type-regex[0]=busnet (osd=1)  
...
[si2oad] Checkpointed at buf loc = 674
[si2oad]     Erased output to checkpoint = 674
[si2oad]     Skip NAME=busNetA1: no match to name-regex[0]=busNet[A-Z][0-9]\[.*:20:[13579]\] (osd=1)
[si2oad] ...MATCH TYPE=BusNetDef to type-regex[0]=busnet (osd=1)
[si2oad] Checkpointed at buf loc = -1
[si2oad]     Erased output to checkpoint = -1
[si2oad]     Skip NAME=busNetA2: no match to name-regex[0]=busNet[A-Z][0-9]\[.*:20:[13579]\] (osd=1)
[si2oad] ...SKIP TYPE=BusTermDef: NO MATCH to type-regex[0]=busnet (osd=1)  
...
[si2oad] ...MATCH TYPE=BusNet to type-regex[0]=busnet (osd=1)
[si2oad] Checkpointed at buf loc = 85
[si2oad]     Erased output to checkpoint = 85
[si2oad] ...MATCH TYPE=BusNet to type-regex[0]=busnet (osd=1)
[si2oad] Checkpointed at buf loc = -1
[si2oad]     MATCH NAME=busNetA2[3:20:5]: name-regex[0]=busNet[A-Z][0-9]\[.*:20:[13579]\] (osd=1)
[si2oad] ...DISPLAY ALL remaining types in container: Past regex nEntries
[si2oad] Checkpointed at buf loc = 826
[si2oad] ...SKIP TYPE=ScalarNet: NO MATCH to type-regex[0]=busnet (osd=1)
[si2oad] ...SKIP TYPE=ScalarNet: NO MATCH to type-regex[0]=busnet (osd=1)

6.4 Examples

The following examples start after running the following:

    cd $Si2oadINSTALLdir/si2distest
    make oad TEST=test1
    b 286
    c

The .si2oadoptions file settings are:

    casesensitive  t  # Default case-sensitivity setting for regex
    regexsticky    f  # If t && regex set, it applies to all subsequent oad()
The verbose option is set to 2 to show extra information during processing.

```plaintext
(gdb) call oad("o", "verbose=2")
(gdb) call oad("x", " lib design block net")
[sii2oad] Processing regex
[sii2oad] ...No case override since no leading + or -
[sii2oad] ...Deleted all prior regex entries
[sii2oad] ...lib matches: Lib AnalysisLib LibDefList LibDef LibDefListRef LibDMData
[sii2oad] ...design matches: Design DesignInst
[sii2oad] ...block matches: Block BlockBoundary AreaBlockage LayerBlockage
```

In setting the regex, note that the first component, lib, matches both Lib and AnalysisLib, while "block" matches any Type name with the word "block" in it. Prefixing a ^ and postfixing a $ to a Type name limits the match to Type names that start and end, respectively, with the indicated string. For example,

```plaintext
(gdb) call oad("x", " ^lib$ design$ block$ net$")
[sii2oad] Processing regex
[sii2oad] ...No case override since no leading + or -
[sii2oad] ...Deleted all prior regex entries
[sii2oad] ...^lib$ matches: Lib
[sii2oad] ...design$ matches: Design
[sii2oad] ...block$ matches: Block
```

The next call to oad() will show all Nets in the current RTM, along with their parent Objects up to the root Object in the call (or the top of the RTM if no root Object is used as an argument). All the attributes of the container Objects will display, but no other owned Objects besides Nets. However, all Objects continuing down the owner hierarchy from each Net owner will display. Single-line Collection indicators for all skipped Object collections will be displayed, along with the number of Objects in the Collection (though these are not expandable to show those Objects).

Since no name expressions were appended to any of the Type names, all names are matched for each of the matched Types. For example, only busNetA2[3:20:5] would be matched by the following regex,

```plaintext
(gdb) call oad("x", " ^s  block   net=busnet.*\[[0-9]*:[13579]\]$")
```

In the call above, a root object was provided; hence the prior regex setting needed to start with the next Type of Object under that root (a Block in this case, since only Block Domain BusNets were to be matched).
6.5 Per-Collection Regex and Sorting

When using the GUI with maxownerlevels=1, which allows loading of only the top Collection header before any of its contents are traversed, it is possible to set different regex and/or sort criteria for each such Collection—regardless of whether a "global" typeRegex/nameRegex pathExpression has been set.

This is accomplished by holding the CONTROL key down when clicking either the + or - on the Collection line (depending upon whether the Collection is currently closed or open). A pop-up dialog solicits the typeRegex, nameRegex and/or sort to be used for this Collection. On successful parse of any regex expressions, only Objects of the matched Types and names will be displayed in the Collection as it is incrementally perused. If sort was selected, the Objects appear in the display ordered according to the algorithm of the sort DLL (7.4) selected. Any combination of typeRegex, nameRegex, and sort may be applied repeatedly to the same Collection.

Figure 3 illustrates the use of the pop-up box to select only those constraintDefs whose names begin with “min” and display them in alphabetical order. The pop-up box appears when the leading – character (or the + if the node is folded) on the constraintDef line is control-clicked:

![Figure 3: Sort/Regex Pop-up](image)

After SUBMIT is clicked, the constraintDefs are selected and sorted:

![Figure 4: Sorted ConstraintDefs](image)
6.5.1 Pop-Up Reuse

The name of the Collection that was CTL-clicked is inserted into the “APPLY TO” (a.k.a., “submit”) button of the pop-up to help keep track of the GUI node the pop-up is affecting (Figure 5). If a bad regex is inserted, the error message resulting from the attempt to compile it will be displayed in the status bar at the bottom of the window frame. The regex can be re-edited in place and tried again. In fact, the pop-up will remain open until some other click action is made in the window, at which time the popup will be automatically closed. The popup will even remain open (for subsequent analysis) if the “RETURN TO CONSOLE” button is pressed. However, as soon as the main GUI window is closed, the pop-up will be automatically closed.

![Figure 5: Regex/Sort Pop-Up](http://localhost:25864 - Mozilla Firefox)

6.5.2 Overhead

The sort and re-regex features require caching of a Collection's members. For large Designs this can add up to a substantial amount of extra memory allocation overhead in the application process. Only incremental Collections for which a CTL-click has been submitted will incur this overhead. Hence, if these features are not used, no memory penalty results.
7 User Extensions

The native Si2OADebug code is extendable in a few, limited ways. User-defined viewers for byte-type Prop or AppDef data, and CMaps, as well as Observer implementation code can be hooked into the oad library at run-time. Sort routines for Object Collections can be defined, as well.

7.1 Prop/AppDef Data

AppProp, DataAppDef, and VarDataAppDef values can be arbitrarily long. Mindlessly displaying them on the console or GUI could cause an excess of output. The maxextdata option sets a limit on how much of this data to show. A setting < 1 will force all data to be displayed. Any positive value will show only that number of bytes with a trailing "..." sentinel to indicate the rest of the data was truncated in the display. In GUI interactive mode, clicking on these dots will cause a callback for the full wad of data to be displayed in a separate window.

- If the dlldatadisp option is not "0", it must be the path of a DLL to be loaded. If the first character is a "$" the rest is interpreted as an environment variable, which will be accessed for the actual path. That DLL must implement a function with "C" linkage, which will be called to display truncated data as. The declaration for this function is as follows, with 0 to be returned on successful display, and any other return value fed back to the oad user via log message:

  
  ```
  int oadViewData( oaByte *val, oaUInt4 size );
  ```

- If the dlldatadisp option = "0", then a click on truncated data sentinel will cause the data to be written to a file and the path specified by the sysdatadisp option will be executed in a system() call, passing to it the name of that file as an argument. Since the dlldatadisp DLL is always checked first, if it is set, even if sysdatadisp is also set, it is ignored.

The DLL is loaded as a result initial setting or changing of dlldatadisp option. Any DLL already loaded as a result of this option setting is first unloaded, and the current value of the option is used as a path to load a new DLL, called in subsequent requests to display truncated extension data.

7.2 CMap

Currently, there is no way to get CMap data into console [file] output. The cmap attribute, essentially a sentinel for the CMap data itself, is always displayed with the value, "...". There is no public API method for uncovering which layerNums have CMap data. This must be specified by the user and currently can only be accomplished via input to a javascript prompt when the sentinel is clicked. The layerNum solicited by this dialog is passed via callback to a either a DLL or system executable for display in a separate window.

- If the dllcmapdisp option is not "0", it must be the path of a DLL to be loaded. If the first character is a "$" the rest is interpreted as an environment variable, which will be accessed for the actual path. That DLL must implement a function with "C" linkage, which will be called to display CMap data for the specified layerNum. The declaration for this function is as follows, with 0 to be returned on successful display, and any other return value fed back to the oad user via log message:
int oadViewCMap( oaCMap *cmap, oaLayerNum layerNum );

• If the dllcmapdisp option = "0", then a click on truncated data sentinel will cause the data to be written to a file and the path specified by the syscmapdisp option will be executed in a system() call, passing to it the name of that file as an argument. Since the dllcmapdisp DLL is always checked first, if it is set, even if syscmapdisp is also set, it is ignored.

The DLL is loaded as a result initial setting or changing of dlldatadisp option. Any DLL already loaded as a result of this option setting is first unloaded, and the current value of the option is used as a path to load a new DLL, called in subsequent requests to display CMap data.

7.3 Observers

Observers can be linked with an application to create useful trace output even without ever calling an oad() function to dump data. Alternatively, the observer implementation code itself might include oad() calls to display selected Objects. The implementation is arbitrary and could even include code that changes the RTM, though that would be at odds with the overall spirit of oad as a diagnostic, rather than editing tool.

Note that traversing the model as a result of an oad() call will cause firing of the relevant Observers injected via this technique.

The si2dis/si2oadObserve.cpp file has a skeleton implementation of an Observer that can be used to track events as the application runs. The file that implements the Observer code can have any name; however, it must be linked as a shared object (DLL). If the observer option is set to the path of this DLL, then Si2OA Debug will open it as the application is loaded into memory.

7.3.1 Declarations

The declarations for the Observer variables should be global (static) in that compilation module (or otherwise allocated as a result of initialization) to give the Observers a chance to be registered as the DLL is loaded.

• If declared in static scope, the Observers will be unregistered automatically when the Si2OA Debug option is set to 0 and the DLL is unloaded.

• If the Observer variables are allocated, then they will continue to be registered even though the DLL is unloaded, unless the Observer class destructor explicitly frees them.

7.3.2 Events

Most oaObjects have standard notifications that will only fire if the Object is created/destroyed/modified — meaningful if Si2OA Debug is linked with an application that performs such actions. Observers for Objects that have postOpen, preDestroy, or similar events, will fire simply as a result of Si2OA Debug traversal activity, even with the stand-alone persistent data display tool.
7.3.3 Example

The following console output illustrates selective load/register and unload/unregister of an Observers DLL to capture/ignore the Prop create events of the application (with some lines omitted and replaced by "..."):

```
[q@localhost si2distest]$ make gdb
...
Breakpoint 1 at 0x8063039: file testcase.cpp, line 32.
No observers library was loaded.
Breakpoint 1, main (argc=1, argv=0xbfffaf4c) at testcase.cpp:32
32 if ( argc > 1 ) {
  (gdb) call oad("w")                 <-------- Export current options settings
Write option settings to replace file path/name [or ? to cancel] oad.conf
    (Point option 26 to a DLL that implements desired Observer code)
  (gdb) shell sed 's;26.*;26 ../si2dis/si2oadObserve.so;' oad.conf >oadObs.conf
  (gdb) call oad("r")                 <-------- Load the changed options settings
Read in option settings from file path/name [or ? for NONE] oadObs.conf
Opening ../si2dis/si2oadObserve.so        <-------- User Observer code being loaded
Constructing Observer<oaProp> priority=4    <--- Observers being registered
Loaded dllObservers OK.
  (gdb) b 59
  c
Breakpoint 2 at 0x8063280: file testcase.cpp, line 59.
  (gdb) Continuing.
onPostCreate: propName=iprop1mod    <-------- Observers firing
onPostCreate: propName=iprop2mod
onPostCreate: propName=iprop1block
Breakpoint 2, main (argc=1, argv=0xbfffaf4c) at testcase.cpp:59
59    *iProp2block = oaIntProp::create(block, "iprop2block", 72 ),
  (gdb) call oad("r")                 <-------- Reload options file prior to change
Read in option settings from file path/name [or ? for NONE] oad.conf
Closing observers library ...          <-------- User Observer code being unloaded
Destructing Observer<oaProp>            <-------- Observers being unregistered
  (gdb) n                                <-------- No more Observer firing
60    *iProp2ablock = oaIntProp::create(iProp2block, "iprop2ablock", 752 ),
  (gdb) n                                <-------- No more Observer firing
61    *iProp2a1block = oaIntProp::create(iProp2ablock, "iprop2ablock", 725 );
  (gdb) call oad(iProp2ablock)
<oaIntProp id='b75d0296' name='iprop2ablock' valueAsStr='752' value='752'/>
```

7.4 Sort DLLs

In GUI mode, CONTROL-click on the leading + or – of a Collection entry will pop up a dialog box to solicit not only a regex filter (6.5) but a <SELECT> list from which a sort routine can be selected to present the Objects in a particular order. Collections can be sorted and resorted according to different criteria, and in combination with different regex filters selected in the same pop-up.

Each of the sort methods listed in the SELECT box corresponds to a DLL defined with a single entry point with the prototype:

```c
void sortObjects( oaObject **objArray,
                  int nObjs )
```
The DLL must be declared with `extern "C"` linkage since `dlopen()` loads it. The DLL may use any method to organize the array of Objects in the Collection to be sorted. The following example sort DLLs are included in the distribution as examples:

- `si2oadSortName.cpp` uses Object names as the sort key. It reuses the `getNameFromObject()` method and so must include “si2oadglob.h”.
- `si2oadSortOaTypeName.cpp` uses `oaType.getName()` values as sort keys.
- `si2oadSortInfoType.cpp` uses the “information” type of the Object as a sort key – an arbitrary designation that means TermType for Terms and SigType for Nets (and is ignored for other Types).

The paths to these are listed (a complete path name could be included if they were not in the current directory), separated by commas, as the value of the `dllsorters` option. Each time the options file is read, the template for the `regexChoices.html` popup dialog is recreated in `outputpath` filled in with the current list of these sort DLLs.

Any number of custom sort DLLs can be loaded dynamically to a running oad process by setting (or changing) the `dllsorters` option in the `.si2oadoptions` file either in advance of an oad session or dynamically when control has returned to the command line.
8 Implementation Details

Understanding Si2OADebug's design criteria and how it works can helpful when trying to solve problems using the tool, cope with its limitations, or customize it for different site configurations and environments.

8.1 Requirements

Practical use case issues drove both the management of the tool development activities and the technical details of the implementation.

8.1.1 Project

For consistency with the spirit of OpenEDA in general, efforts have been made to

- Encourage participation across the OA Community
  - Close leadership by the WG has helped broaden requirement inputs and testing.
  - Si2 development of the prototype under WG guidance minimized IP encumbrances while providing a vehicle for distributing the support burden.
  - Making source available will (hopefully) provide a vehicle and the means for on-going support.
  - Staging the WG activities under an OpenEDA project umbrella, with accompanying flyers, as well as on-line, and in-person demos for the tool in various stages, has helped stimulate broader interest and cooperation from more member companies.

- Minimize development time and cost.

  To support evolution of requirements driven by WG feedback, rapid prototyping of new features was desired. This drove reuse of existing technology wherever possible:
  - XML, a mature cross-industry syntax with extensive cross-industry support to leverage of existing and free parsing and rendition tools.
  - Publicly available browsers for interactive display (as opposed to a custom GUI)
  - XSL to bridge the gap between XML and browser HTML
  - Javascript to provide active an interface

8.1.2 Technical

Specific use case goals drove development of features to

- Enable developers to "see" design data easily to help spot tool bugs and database problems.

  This led to support for a human-readable representation suitable for both batch and GUI uses, a format that lent itself to easy assimilation, with a structure that mapped closely to the RTM in which an OA programmer has to live.
• Allow use of the tool with existing application executables.

Commercial tools are typically not available in source or as unlinked binaries. The LD_PRELOAD (8.2.1) technique makes it possible to debug using the OA data, even without access to the tool code. In addition, shared object "bundling" (8.2.2) has also been successfully used (which would work on platforms that do not support LD_PRELOAD).

• Avoid altering the RTM.

Bounding Box calculations can be switched off. States of master relationship binding, as well as parasitics loading, are preserved, and selectively processed dynamically on a temporary basis. Option settings also allow such RTM changes on an automated basis.

• Minimize impact on memory allocation and performance.

As much as possible, the code attempts to pass model information directly from traversal to output, without requiring large caches of intermediate data and look-ahead processing. Regex processing, however, breaks this goal because it is not possible to know in advance if data down in an owner hierarchy will be selected until it is accessed.

• Support both batch export and interactive, point-and-click renditions.

• Use the public API to insulate against cross-release changes to the OA Reference Implementation.

The public API has been used for all design data access except in two cases:

♦ AppDef access exploits API class implementation knowledge that all subtype AppDefs can be accessed directly from the underlying class oAAppDef. An oAdAppDef class was derived directly from oAAppDef to get access to these private accessors. This enables processing of both Session and Object AppDefs with a single function of some 150 lines. Using only the public API would require a large number of templates be explicitly declared (one for each possible combination of Object Type and AppDef subtype) since it cannot be known until run-time exactly which combinations will be encountered in the data. This technique will break if the OA Reference Implementation ever implements an explicit accessor in an AppDef child class.

♦ Iter caching (to support incremental loading of Collections, a few Objects at a time) exploits the knowledge that the API implementation uses a common oABaseIter class destructor. Using only public API methods would require coping explicitly with every possible template type of Iter that could be involved in user data Collections. This technique will break if the OA Reference Implementation ever implements an explicit destructor in an Iter child class.

Other technical requirements were considered and even investigated, but ultimately discarded. For example, automating the generation of the OAD code was initially deemed desirable. In fact, some legwork was performed studying automated code generation from a model, for example, with a commercial UML tool (like Rational Rose or Argo). However, first a complete UML model, along with all its constraints, would have to be drawn – itself a reasonably time-consuming task. Since such automated generation often produces a clunky result, significant tweaking of the output was likely to be needed. Though such a UML model would undoubtedly be reusable, since it was not clear that OAD would even be that useful to OA developers and engineers, the WG did not want to undertake a significant effort as a prerequisite to even the first line of useful debugging code.
8.2 Linking to an Executable

There are a couple of techniques for making the Si2OADebug symbols (produced by the compiler) available to a debugger so it can call the oad interface functions – even though that library was not linked in when the application to be debugged was created.

8.2.1 LD_PRELOAD

The current technique implemented in the `make oad` target causes a .gdbinit file to be created that lists for LD_PRELOAD the libsi2oadebug.so and the list of OA shared libraries (either dbg or opt versions, whatever is set in makefile.defs). Note, however, that this technique may not be portable to all platform/compiler combinations.

8.2.2 Bundling Shared Libraries

![Figure 6: Bundling With an OA Shared Library](image)

An alternative technique that may be more portable to other development platforms uses the linker to "bundle" the Si2OADebug library with one of the OA shared libraries – one that is required no matter how limited a tool's use of the API might be, such as liboaCommon.so. The resulting "Trojan horse" version of the OA shared lib could then be located in a different directory from the one holding the OA libs. That directory would be placed prior to the main OA lib/ directory on the LD_LIBRARY_PATH so that the bundled lib is accessed first during the dynamic loading process, thereby pulling in the oad() symbols and causing the necessary static initializations for use by the debugger. For example, 8.2.2 illustrates this concept and the corresponding LD_LIBRARY_PATH would be set to something like, $ToolDir:/OAD/lib:$OA_ROOT/lib. This technique has been tested successfully on Linux and Solaris platforms; however, it is not currently being used in the distribution makefiles, which use the LD_PRELOAD technique (8.2.1).
9 TCL Interface

The si2dis/Makefile that produces libsi2oadebug.so also creates an interface to it callles, si2oadtcl.so. This is created using SWIG.

The si2distest/ runtcl make target shows how to use this code.

At this time, only two of the oad() interface functions are wrapped by the si2oadtcl.i file:

- oad $objectVariable
- oad “command” “options”
10 XML Tools

Some browsers do not include a native XSLT feature, or it does not properly process the XML output of Si2OADebug. For such situations, or for the case the browser XSLT feature works slowly, a separate XSLT processor may be used.

10.1 Si2 Redistributed Components

For convenience, Linux binaries for Xalan and xerces that have been tested with Si2OADebug can be downloaded directly from the OpenEDA website from

https://www.si2.org/openeda.si2.org/project/showfiles.php?group_id=29#p65

The regex and insertDLL libs (described in Section #2.4.1) are also on this site.

10.1.1 Apache Distribution

Alternatively, the packages can downloaded directly from Apache; however, the latest versions likely to be on that site have not been tested with the code.

• Xalan: XSLT processor (C++)

http://www.apache.org/dist/xml/xalan-c/binaries/Xalan-C_1_9_0-redhat_80-gcc_32.tar.gz
http://www.apache.org/dist/xml/xalan-c/binaries/Xalan-C_1_9_0-redhat_80-gcc_32.tar.gz.sig

• xerces: XML parser (C++) used by Xalan

http://archive.apache.org/dist/xml/xerces-c/Xerces-C_2_6_0/xerces-c_2_6_0-redhat_80-gcc_32.tar.gz
http://archive.apache.org/dist/xml/xerces-c/Xerces-C_2_6_0/xerces-c_2_6_0-redhat_80-gcc_32.tar.gz.asc

Signatures on all downloaded packages should be verified. An example procedure for is explained on,

http://xml.apache.org/xerces-c/download.cgi#verify

The following documents the security verification performed on those packages:

[localhost XML]$ gpg --import KEYS.xerces

gpg: WARNING: using insecure memory!
gpg: please see http://www.gnupg.org/faq.html for more information
gpg: key 24E87419: "Tinny Ng (Xerces-C++ committer) <tng@apache.org>" not changed
gpg: key 24E87419: "Tinny Ng (Xerces-C++ committer) <tng@apache.org>" not changed
gpg: key AF8777A6: "Gareth Reakes <gareth@apache.org>" not changed
gpg: key 95597B05: "Neil Graham (this key is primarily for signing Xerces-J releases) <neilg@ca.ibm.com>" not changed
gpg: key 71DB2335: "PeiYong Zhang <peiyongz@ca.ibm.com>" not changed
gpg: Total number processed: 5
gpg: unchanged: 5

[localhost XML]$ gpg --import KEYS.xalan

gpg: WARNING: using insecure memory!
gpg: please see http://www.gnupg.org/faq.html for more information

gpg: key 30A21D55: public key "Shane Curcuru <shane_curcuru@lotus.com>" imported
gpg: key 6442C3DC: public key "Lotusxsl Team <Lotusxsl_Team@lotus.com>" imported
gpg: key 1AAC221B: public key "Joseph Kesselman <joseph_kesselman@lotus.com>" imported
gpg: key 58DA77EA: public key "David N. Bertoni <david_n_bertoni@us.ibm.com>" imported
gpg: key 55DEED55: public key "Henry Zongaro <zongaro@ca.ibm.com>" imported
gpg: key 2004CC44: public key "Matthew Hoyt <mhoyt@ca.ibm.com>" imported
gpg: key C7599162: public key "Dmitry Hayes <dmitryh@apache.org>" imported

gpg: Total number processed: 7
gpg: imported: 7

[d@localhost XML]$ gpg --verify xerces-c_2_6_0-redhat_80-gcc_32.tar.gz.asc
gpg: WARNING: using insecure memory!
gpg: please see http://www.gnupg.org/faq.html for more information

gpg: Signature made Fri 01 Oct 2004 02:15:13 PM CDT using DSA key ID 71DB2335
| gpg: Good signature from "PeiYong Zhang <peiyongz@ca.ibm.com>"
| gpg: WARNING: This key is not certified with a trusted signature!
| gpg: There is no indication that the signature belongs to the owner.
| Primary key fingerprint: CDFF AFA9 0F21 B79B 8C94 9A1B ECEB 0855 71DB 2335

[d@localhost XML]$ gpg --keyserver pgpkeys.mit.edu --recv-key 71DB2335
| gpg: WARNING: using insecure memory!
| gpg: please see http://www.gnupg.org/faq.html for more information
| gpg: key 71DB2335: "PeiYong Zhang <peiyongz@ca.ibm.com>" not changed
| gpg: Total number processed: 1
| gpg: unchanged: 1

[d@localhost XML]$ gpg --fingerprint 71DB2335
| gpg: WARNING: using insecure memory!
| gpg: please see http://www.gnupg.org/faq.html for more information
| pub 1024D/71DB2335 2004-02-16 PeiYong Zhang <peiyongz@ca.ibm.com>
| Key fingerprint = CDFF AFA9 0F21 B79B 8C94 9A1B ECEB 0855 71DB 2335
sub 1024g/DCAA02DB 2004-02-16

[d@localhost XML]$ gpg Xalan-C_1_9_0-redhat_80-gcc_32.tar.gz.sig
| gpg: WARNING: using insecure memory!
| gpg: please see http://www.gnupg.org/faq.html for more information
| gpg: Signature made Tue 21 Dec 2004 12:59:00 PM CST using DSA key ID C7599162
| gpg: Good signature from "Dmitry Hayes <dmitryh@apache.org>"
| gpg: checking the trustdb
| gpg: no ultimately trusted keys found
| gpg: WARNING: This key is not certified with a trusted signature!
| There is no indication that the signature belongs to the owner.
| Primary key fingerprint: EE6E 40A4 16B0 43DB 18CC 8F3C 3F62 1BF0 C759 9162

[d@localhost XML]$ gpg --keyserver pgpkeys.mit.edu --recv-key C7599162
| gpg: WARNING: using insecure memory!
| gpg: please see http://www.gnupg.org/faq.html for more information
| gpg: key C7599162: "Dmitry Hayes <dmitryh@apache.org>" not changed
| gpg: Total number processed: 1
| gpg: unchanged: 1

[d@localhost XML]$ gpg --fingerprint C7599162
| gpg: WARNING: using insecure memory!
| gpg: please see http://www.gnupg.org/faq.html for more information
| pub 1024D/C7599162 2004-12-21 Dmitry Hayes <dmitryh@apache.org>
| Key fingerprint = EE6E 40A4 16B0 43DB 18CC 8F3C 3F62 1BF0 C759 9162
sub 1024g/ACD96CE4 2004-12-21

(4) Browser
------

http://download.mozilla.org/?product=firefox-1.0.7&os=linux&lang=en-US
11 Problem Resolution

Problem: \texttt{gdb} reports error mapping shared library and subsequent call to \texttt{oad()} SEGVs:

```
(gdb) call oad()
Program received signal SIGSEGV, Segmentation fault.
0x00d8f76c in OpenAccess_4::oaObjectV::get(OpenAccess_4::oaObject const*)
  (object=0xb75d0012) at oaObjectPvt.inl:42
```

Solution: Do not call \texttt{oad()} on an OA object of some kind until the an \texttt{oa*Init()} initialization function – like \texttt{oaDesignInit()} – has been executed by the application being debugged. This SEGV typically occurs when such a call is made immediately on load of the application by \texttt{gdb}, instead of after execution has continued to the first breakpoint. Calling \texttt{oad()} to set its own internal options is possible at any time; however, any attempt to get information on OA data must wait until the OA implementation has been initialized.

Problem: \texttt{gdb} reports error mapping shared library and subsequent call to \texttt{oad()} SEGVs:

```
make gdb
...
Breakpoint 1 at 0x804ad7a: file si2dis.cpp, line 89.
Error while mapping shared library sections:
libsi2oadebug.so: Success.
...
```

Solution: Add to the \texttt{.gdbinit} file a "set environment" statement for \texttt{LD_LIBRARY_PATH}, even though it may already be set in the environment:

```
set environment LD_LIBRARY_PATH .:$OAROOT/lib/linux_rhel21_32/dbg
```

Problem: Soon after starting a testcase in si2distest (or other code) an uncaught exception terminates execution.

```
Breakpoint 1, main (argc=1, argv=0xbfff9ea4) at testcase.cpp:38
38     if ( argc > 1 ) {
(gdb) b 109
Breakpoint 2 at 0x804cf58: file testcase.cpp, line 109.
(gdb) c
Continuing.
***Exception: Library Lib not found.
```

Solution: The testcases (and many programs) depend on a lib.defs file having been set up to map logical Lib to physical path names. If working with a testcase in the distribution simply run the following make target then restart the testcase from the beginning:
Problem: Browser displays raw XML with all of the container tags expanded and, except for the collapse(−)/expand(+) icons, does not have active content. The display appears like that in 11.

Figure 7: XSLT Failure

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<si2oaddebug>
  <oaBusNet name="busNetA1[3:20:2]" id="b75d041c" domain="block" block="#b75d0312" isEmpty="t" numBits="9">
    <start id="3" step="20" step="2">
      <design ref="#b75d0014" lcr="Lib/SampleSchematic"/>
      <occNet ref="#b75d049c" name="busNetA1[3:20:2]"/>
    </start>
    <occNets count="1">
      <oaOccBusNet ref="#b75d049c" name="busNetA1[3:20:2]"/>
    </occNets>
    <singleBitMember count="9">
      <oaBusNetBit ref="#b75d041d" name="busNetA1[3]"/>
      <oaBusNetBit ref="#b75d041e" name="busNetA1[5]"/>
      <oaBusNetBit ref="#b75d041f" name="busNetA1[7]"/>
      <oaBusNetBit ref="#b75d0420" name="busNetA1[9]"/>
      <oaBusNetBit ref="#b75d0421" name="busNetA1[11]"/>
      <oaBusNetBit ref="#b75d0422" name="busNetA1[13]"/>
      <oaBusNetBit ref="#b75d0423" name="busNetA1[15]"/>
      <oaBusNetBit ref="#b75d0424" name="busNetA1[17]"/>
      <oaBusNetBit ref="#b75d0425" name="busNetA1[19]"/>
    </singleBitMember>
    <def ref="#b75d0512" name="busNetA1"/>
  </oaBusNet>
</si2oaddebug>
```
**Solution:** Either "XSLT transformation type" option setting is

- **processor** but either of following options do not reference the correct paths:
  - "Path to the XSLT processor executable" – with an error message on the console like,
    ```
    [si2oad] ***Command failed: LD_PRELOAD="
    LD_LIBRARY_PATH=.:/home/oa/labs/lib:/home/226p052/lib/linux_rhel21_32/dbg:$HOME../XML/lib
    $BAD_PATH/XML/bin/Xalan
    -o /home/si2oad.22805/si2oad.html
    /home/si2oad.22805/si2oad.xml ../si2dis/si2oad.xsl
    ```
  - "Colon-separated paths to libs needed by XSLT"– with an error message on the console like the one above, plus a prior error when loading the libs was attempted, such as,
    ```
    [si2oad] ...Stand-alone XSLT processing starting on /home/si2oad.22805/si2oad.xml
    /home/..XML/bin/Xalan: error while loading shared libraries: libxalan-c.so.19: cannot open shared
    object file: No such file or directory
    ```

- **browser** but either
  - The browser does not have its own XSLT capability. In this case, use a different browser or the **processor** setting.
  - The si2oad.xsl file did not make it into the `docroot/` directory specified by the "Document root path for microwww server" option setting. In this case, fix the makefiles or manually copy the file into the `docroot/`.

**Problem:** Link failure of system library, such as:

```
 Breakpoint 1 at 0x405fa3: file testcase.cpp, line 39.
 /bin/sh: error while loading shared libraries: /lib/libdl.so.2: cannot
 open shared object file: No such file or directory.
```

**Solution:** Default system library paths should include /lib and /usr/lib unless the linker has been configured in a special way. Verify exactly which library resolution the shared object is expecting:

Run `ldd $Si2oaDebug_INSTALL/lib/si2oaddebug.so`:

```
  libstdc++.so.5 => /usr/lib64/libstdc++.so.5
  libm.so.6 => /lib64/tls/libm.so.6
  libgcc_s.so.1 => /lib64/libgcc_s.so.1
  /lib64/ld-linux-x86-64.so.2 => /lib64/ld-linux-x86-64.so.2
  ...
```

If system libraries are being resolved to paths other than the linker's defaults, they can be added explicitly to the `LD_LIBRARY_PATH` lists by editing the `$Si2oaDebug_INSTALL /makefile.defs` for the targets being used. For example, add `/lib64` to to the `oad` target.

```bash
 oad: RUNoad
 RUNoad: $(PROG)
 $(MAKE) gdbinit
 $(SH_LIB_ENV_NAME)=.:.$(DIR_LIB_OAD):$(DIR_LIB_OA):/lib64 \
  PATH=$(PATH):$(DIR_BIN_OA) \
 $(GDB) $(PROG)
```
**Problem:** ***Exception: Unable to lock database file for LibDetPara/FullAdder/routed***

**Solution:** If it is possible that another application is accessing the same Design, then the solution is to wait until the other application is finished. If no other applications are (or should be) accessing that Design, then a crash or some other event may have caused the design Data Manager (DM) to leave a remnant lock for the Design. The exact solution depends on the DM PlugIn used to create the Lib for the Design. For example,

- DMFileSys creates a "stake file" in the directory named in the Exception message with the name of the primary DMFile representing the Design with suffix, ".cdslck". This file can be manually deleted to release the lock.

- DMTurbo may leave a remnant server process active that needs to be killed and/or a stake file that must be deleted. First try deleting the stake file.

  ```
  (gdb) shell rm $(LIBLOC)/LibDir/LibName/server.xml
  ```

  If that is not sufficient, then locate the TurboServer process and kill that. If all else fails, recreate the Lib data.

  ```
  (gdb) shell ps -ae | grep DMTurboServer
  8019   ?  00:00:00   oaDMTurboServer
  (gdb) shell kill -9 8019
  ```

**Problem:** Debug shell process hangs without returning to a prompt even though the RETURN button was pressed (or “quit” URL accessed) to terminate the browser communications session.

**Solution:** If an instance of Firefox was not running when the Si2oaDebug created it, the process is waiting on its children to terminate. Next time, start a Firefox first before running OAD.
12 Known Defects

12.1 Missing Class Implementations

The following managed Objects have not yet been implemented:

- WaferDesc
- Frame
- Reticle
- ReticleRef
- WaferFeature
- StepperMap
- Image
- FrameInst
- DesignInst

12.2 Miscellaneous Problems

**Problem:** The internal XSLT of Firefox 1.5.0.1 eliminates all newlines.

**Workaround:** The Firefox 1.0 "preview" version (and maybe some other version) works fine, as does Mozilla 1.4. Alternatively, use an external XSLT processor (like the supplied Xalan) to create HTML. Set option `xsltype` to ‘processor” and use Firefox to display the HTML produced in that fashion.

**Problem:** The OA model is incomplete.

**Workaround:** All public classes and attributes will eventually be implemented in subsequent releases of Si2OADebug. Meanwhile, to satisfy critical needs, the source can be edited as needed to add new attribute calls missing from classes or new classes that have not yet been implemented. The makefiles will automatically rebuild the application as needed. As an alternative, see the last “Problem” in this section.

**Problem:** The ______ [fill in the blank] feature does not work the way it should (i.e., the way I need).

**Solutions:**

- Log an bug or enhancement request in the formal tracker for consideration by the Working Group: [https://www.si2.org/openeda.si2.org/tracker/?group_id=29](https://www.si2.org/openeda.si2.org/tracker/?group_id=29)
- Join the Debug Working Group to help influence the direction of development: [http://www.si2.org/?page=667](http://www.si2.org/?page=667)