SANDIA REPORT
SAND2011-4129
Unlimited Release
Printed June 2011

JAGUAR DEVELOPER'S MANUAL

Ethan Chan

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2011-4129
Unlimited Release
Printed June 2011

JAGUAR DEVELOPER'S MANUAL

Ethan Chan

Quantitative Modeling and Analysis
Sandia National Laboratories
P.O. Box 969
Livermore, CA 94551-MS9155

Abstract

JAGUAR (JAva GUi for Applied Research) is a Java software tool providing an advanced text editor and graphical user interface (GUI) to manipulate DAKOTA (Design Analysis Kit for Optimization and Terascale Applications) input specifications. This document focuses on the technical background necessary for a developer to understand JAGUAR.

Contents

Chapter 1 INTRODUCTION

Welcome to the JAGUAR developer guide. We will go through the underlying architecture so you can better understand how JAGUAR was built and how it works.

Section 1.1 PACKAGE OVERVIEW

JAGUAR 2.1 has been developed using Eclipse Helios 3.6.1, Windows 7 64-bit, JDK 6 64-bit. Other configurations may also be compatible.

JAGUAR requires two projects to be checked out from the SVN repository located at **https://dta.ran.sandia.gov/svn/dart/DART-WB-Core/trunk/**

1. **/gov.sandia.dart.jaguar** - The main project where JAGUAR lives.
2. **/gov.sandia.dart.jaguar.feature -** Used to build JAGUAR for different platforms.

Section 1.2 PACKAGE HIERARCHY

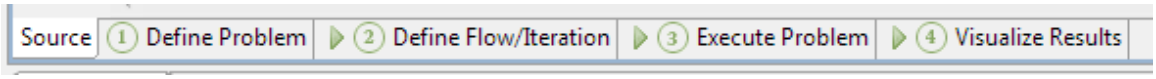
The basic hierarchy structure:

- **/gov.sandia.dart.jaguar/**
  - **src/gov/sandia/dart/jaguar/**
    - **core**
      - **form –** graphical renderers of the internal model
      - **model –** components that build up the model representation
      - **parse –** better parse text into syntactical objects
    - **editors**
      - **application –** overrides menus/look & feel etc. A lot of overrides.
      - **partition2 –** Text editor core
    - **intro –** The welcome page
    - **wizard**
      - **LHS –** Latin Hypercube Sampling (aka Sensitivity Analysis)
      - **newInputDeck –** Create new inputdeck

        - **newInputDeckFromTemplate –** Create new input file from user created and JAGUAR standard templates.
    - **cheatsheet –** cheatsheet definitions
    - **doc –** JAGUAR User's Manual / documentation
    - **files –** dakreord, templates, dakota desc/nspec files
    - **icons –** the icon sets used in JAGUAR
    - **sqa** – released process and test process notes
    - *jaguar.product* **–** configurations used to build the binaries
    - *plugin.xml* **–** defines the dependencies and plugin specifics
- **/gov.sandia.dart.jaguar.feature/**
    - *build.properties* – controls setting the permissions (specifically for Linux) and copying platform specific files to the generated binary.
    - *COPYRIGHT, LICENSE & README* **–** text files included in the binary.
    - *Dakreorder, icons (and JRE (but unused))* – other necessary files included in the binary.

Important files:

- **editors/JaguarEditor.java**
    - Dynamically generates *NIDR_guikeywds.h* (used for grammar generation) from internal files: dakota.input.desc and dakota.input.nspec
    - Creates the instances of the 5 tabs *(see addPages())*



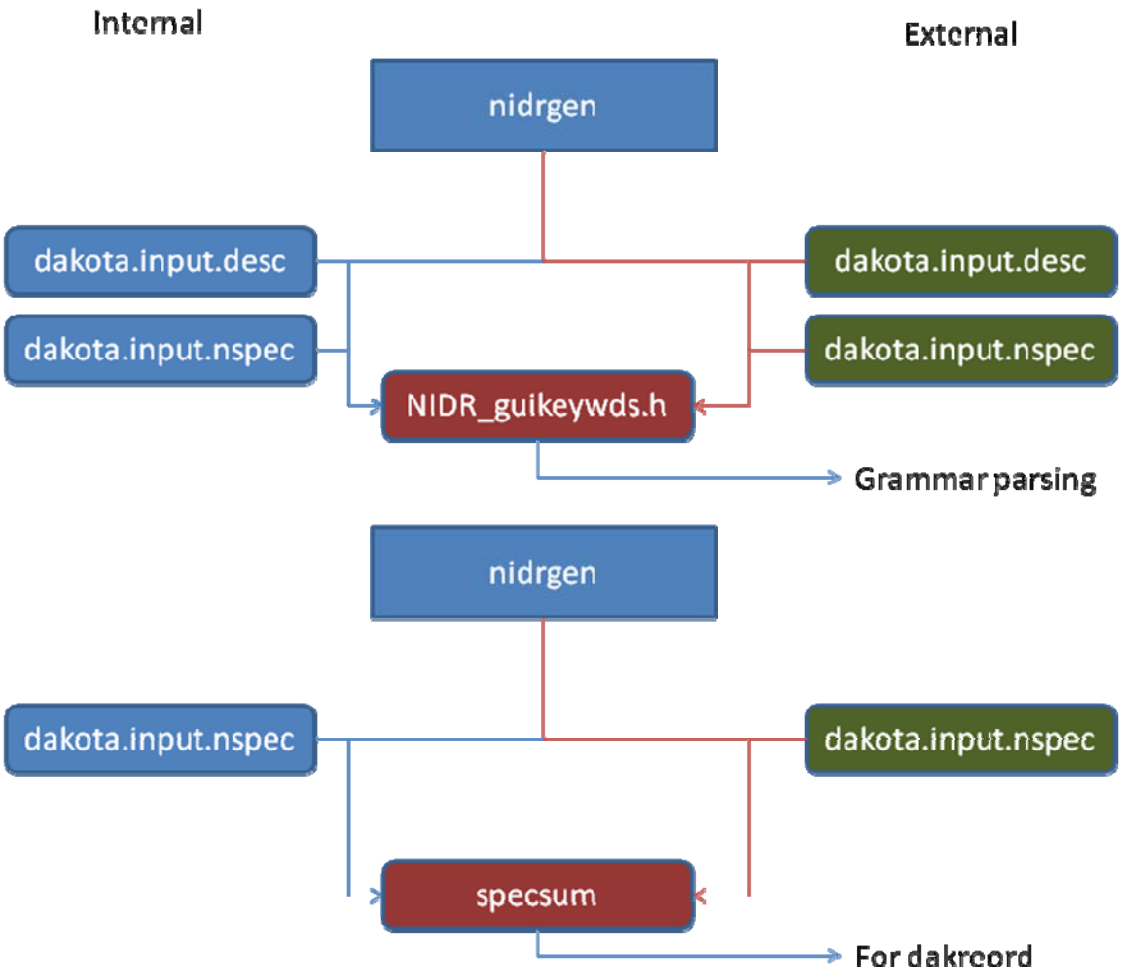
    - Creates instance of JaguarTextEditor
- **editors/partition2/JaguarSourceViewerConfiguration.java**
    - Text editor magic: autocomplete, hover tool tip, and hyperlink
- **editors/partition2/JaguarTextEditor.java**
    - Highlights element at cursor position (see *updateHighlight())*
    - SaveAs & SaveAsTemplate override
    - Updates model and error from text (*see update()*)

Section 1.3 READING IN GRAMMAR

The grammar JAGUAR uses is a direct transfer from DAKOTA. This helps keep JAGUAR always up to date with new DAKOTA releases. A key feature of JAGUAR is to dynamically pull the grammar in and process the input deck according to the grammar specifications.

Relevant code can be found at:
/gov.sandia.dart.jaguar/src/gov/sandia/dart/jaguar/editors/JaguarEditor.java

In method: ***getGrammarModelNIDR()***



- Checks if *dakota.input.desc* and *dakota.input.nspec* exists in user given external Dakota path, otherwise will use internal version.
    - Internal versions located at */files/dakota.input.desc* and */files/dakota.input.nspec*
- Run nidrgen to generate guikeywds (used for reading in grammar)
    - *nidrgen -egG dakota.input.nspec dakota.input.desc*
    - stdout pipes into NIDR_guikeywds.h (located in
- Generate specsum (used for dakreorder future runs):
    - *nidrgen –jspecsum dakota.input.nspec*

Section 1.4 DISSECTING THE GRAMMAR FILE

The NIDR grammar file looks something like this:

```
kw_1[3] = {
      {"active_set_vector",8,0,1,0,1567},
      {"evaluation_cache",8,0,2,0,1569},
      {"restart_file",8,0,3,0,1571}
      },
kw_2[1] = {
      {"processors_per_analysis",9,0,1,0,1551,0,0.,0.,0.,0,"{Number
of processors per analysis}
http://www.cs.sandia.gov/dakota/licensing/votd/html-
ref/InterfCommands.html#InterfApplicDF"}
      },
kw_3[4] = {
      {"abort",8,0,1,1,1557,0,0.,0.,0.,0,"@[CHOOSE failure
mitigation]"},
      {"continuation",8,0,1,1,1563},
      {"recover",14,0,1,1,1561},
      {"retry",9,0,1,1,1559}
      }
```

To decode the grammar, we go through each parameter (not all parameters need to be defined):

Parameters:

1: **keyword name –** The official name of this keyword

2: **kind –** The integer value of *kind* is the summation of the binary values of the below representations:

		512	256	128	64	32	16	8	4	2	1
Integer	**0x1**									**0**	**1**
Real	**0x2**									**1**	**0**
String	**0x3**									**1**	**1**
Array	**0x4**								**1**		
Primary	**0x8**							**1**			
StrictLB	**0x10**					**0**	**1**				
CaneqLB	**0x20**					**1**	**0**				
LB	**0x30**					**1**	**1**				
StrictUB	**0x40**			**0**	**1**						
CaneqUB	**0x80**			**1**	**0**						
UB	**0xc0**			**1**	**1**						
TopLevelAtMostOnce	**0x100**	**0**	**1**								
TopLevelOnlyOnce	**0x200**	**1**	**0**								
TopLevelAtLeastOnce	**0x300**	**1**	**1**								

Meaning:

- **Integer, Real, String** – Each keyword can be of anyone of these types of value, or not specified at all (meaning no value).
- **Array** – Applied for keywords of type integer, real or string representing a list of values (i.e. 1.4,2.5,3.6,4.7 or ‘a’,’b’,’y’, ‘z’)
- **Primary** – If false, represents this keyword is instead an alias. Only primary keywords are shown in JAGUAR GUI.
- **StrictLB, CaneqLB, LB** – Represents lower bound (<, <=) limits exist. See bound values in later parameter definitions. Currently not fully utilized in JAGUAR.
- **StrictUB, CaneqUB, UB** – Represents upper bound (>, >=) limits exist. See bound values in later parameter definitions. Currently not fully utilized in JAGUAR.

- **TopLevelAtMostOnce** – Only applies for section keywords. Indicates only allows 0 or 1 instances (at most once).
- **TopLevelOnlyOnce** – Only applies for section keywords. Indicates only allows 1 instances (only once).
- **TopLevelAtLeastOnce** – Only applies for section keywords. Indicates only allows 1 or more instances (at least once).

For example, 0x7 would represent the keyword as an array of strings -- String (0x3) and Array (0x4).

For the developer's convenience, in ModelElement.java, there are methods prepended with "isKind" such as *isKindPrimary()* and *isKindArray()* used to query these values.

3: **Number of keywords –** Number of keywords contained in the pointer group.

4: **Alternate Group –** Specifies which ModelGroup in the ModelContainer this element belongs to. This will not be referenced directly.

5: **Required –** Determines if this keyword is required to be valid

6: **Alias Group –** not used

7: **Pointer Name –** Refers to the keyword group (kw_#) containing the children keywords.

8: **Lower bound –** defines lower bound (for numbers)

9: **Upper bound –** defines upper bound (for numbers)

10: **Default Real –** default number value (despite the internal name, it is for both real and integer values).

11: **Default String –** default string value.

12: **Description**

```
{"num_least_squares_terms",0x29,6,3,1,1627,kw_173,0.,0.,0.,0,"[choose
response type]{{Least squares (calibration)} Number of least squares terms}
http://www.webaddress.com#RespFnLS"}
```

Default in Description: begin description with@: this specifies if this is in a group that it is the default

Header in Description: contained in between [ ]

Pretty subgroup: contained in between { }

{{A}B}C : A = pretty Subgroup name, B = prettyName, C=rest of description

{D}E: D = prettyName, E = rest of description

13: **Group description**

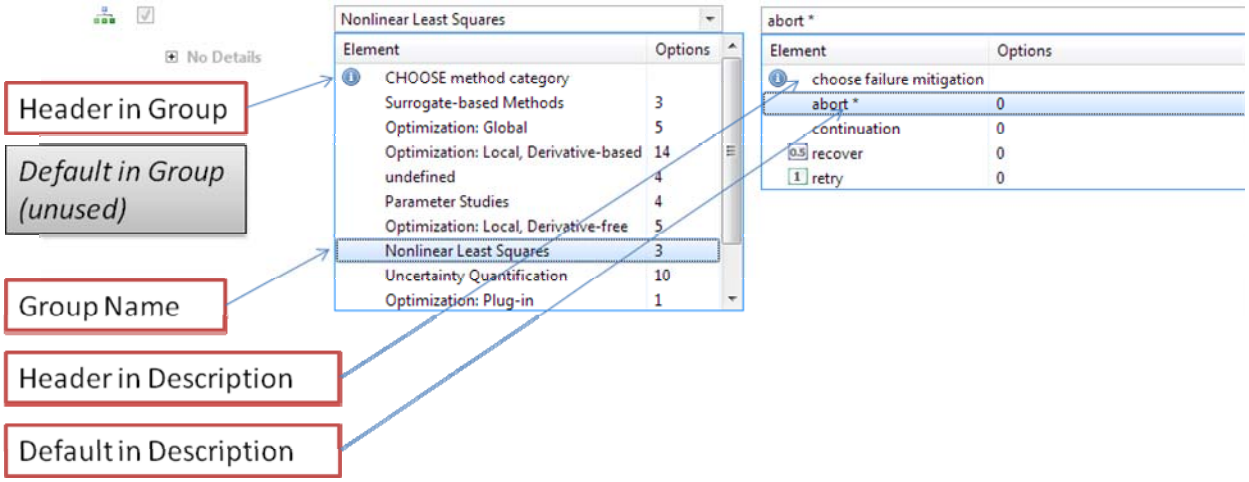
```
{"conmin_frcg",8,9,11,1,177,kw_31,0.,0.,0.,0,"[CHOOSE OPT
method]","Optimization: Local, Derivative-based"}
```

Default in Group: begin description with @: this specifies it is the default

Header in Group: contained in between [ ]

Group Name: rest of description

To clarify parameters 12 and 13, this is how they are used in the GUI:

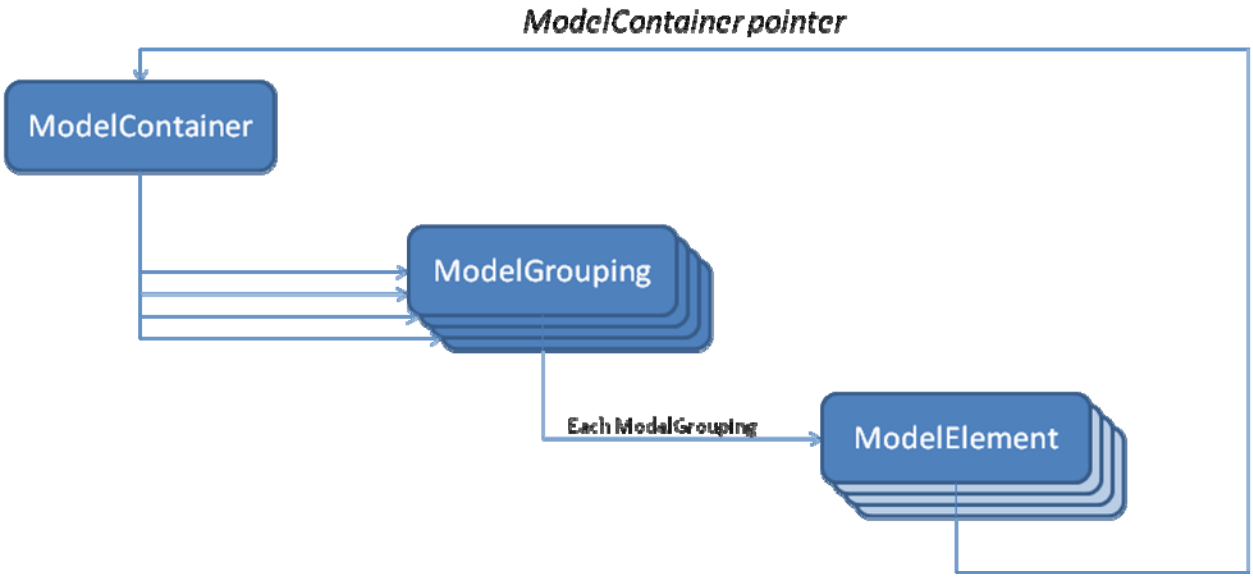


Chapter 2 TEXT EDITOR

Section 2.1 OVERVIEW

JAGUAR uses the generated guikeywds file as the DAKOTA grammar to understand the input deck and to generate the graphical grammar interface.

The grammar is structured in a specific way which can be modeled this way:

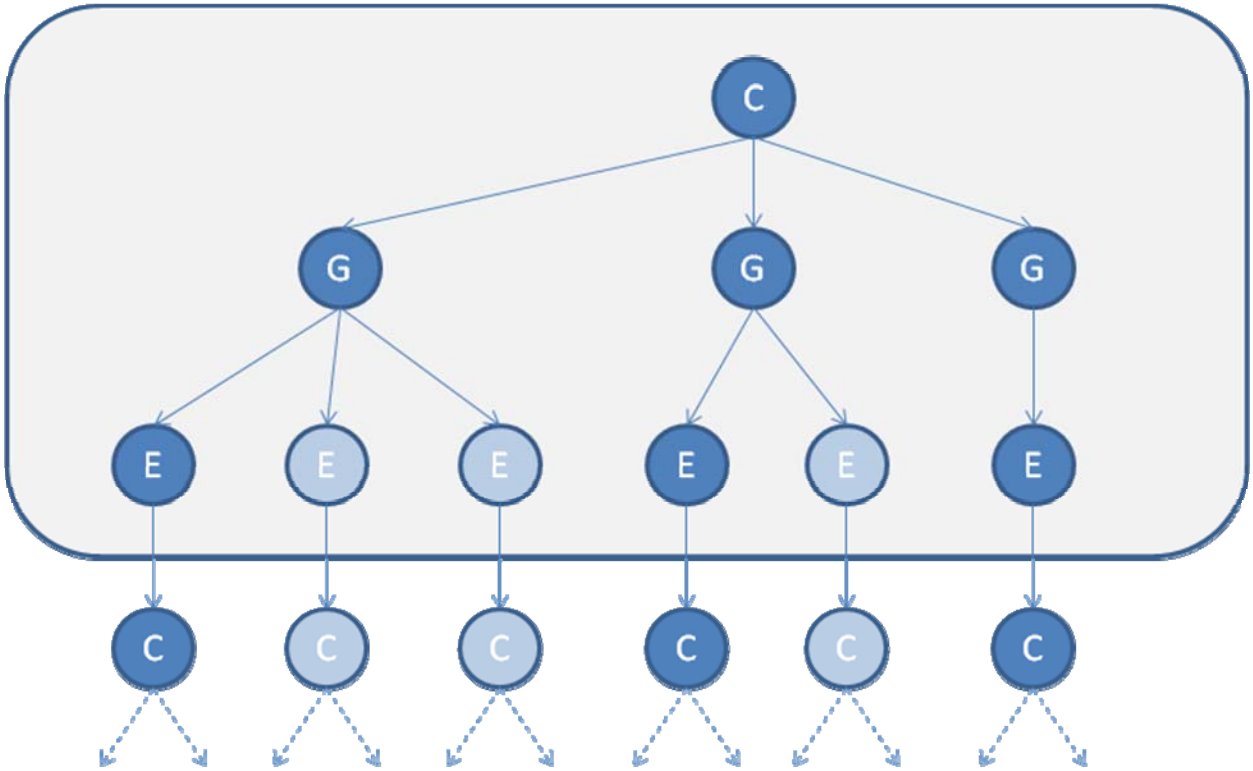


ModelElement – the most important part of the grammar, containing essential metadata (name, value type, limits, prettyName, pointer to ModelContainer etc.). The keywords in the grammar are ModelElements. The pointer to a ModelContainer allows nested behaviors

ModelGrouping – contains a list of ModelElement, but only one of them can be chosen. In essence, a ModelGrouping can be viewed as a combo box, where only one item in the group can be selected.

ModelContainer – contains a list of ModelGrouping, which each has at most one ModelElement selected. This can be viewed as a list

This can be viewed in this way:



This is a view of a single ModelContainer node. In this example it has 3 ModelGroupings. ModelGroupings by themselves are not useful, but each needs to have one of the ModelElements selected. Notice it is often the case where a ModelGrouping only has one ModelElement in it. Remember, the sole purpose of ModelGrouping is to have one selected from possibly multiple ModelElement options.

Let's explore the elements from this sample input deck:

```
strategy
    single_method
        method_pointer 'what'
```

strategy is a ModelElement. Let's take a look at it through a debug run:

this	ModelElement (id=351)
aliasGroup	1
arrayLengthContainer	null
beforeComment	"" (id=366)
changes	null
defaultInDescription	false
defaultInGroup	false
defaultReal	0.0
defaultString	"" (id=366)
description	" The strategy specifies the top level technique w...
elementKeywordLength	-1
elementKeywordOffset	-1
elementValueOffset	-1
grouping	1
groupName	null
headerInDescription	null
headerInGroup	null
inGUI	true
inlineComment	"" (id=366)
instance_enabled	false
instance_value	null
kind	264
lowerBound	0.0
name	"strategy" (id=353)
numElements	10
parentGroupingReference	ModelGrouping (id=368)
pausePropChange	false
pointer	ModelContainer (id=378)
prettyName	"Strategy" (id=380)
prettySubgroupName	null
required	1
topLevel	true
upperBound	0.0

Notice it has a pointer to a ModelContainer. That ModelContainer has a list of ModelGrouping:

pointer	ModelContainer (id=378)
elementData	Object[10] (id=383)
[0]	ModelGrouping (id=390)
[1]	ModelGrouping (id=391)
[2]	ModelGrouping (id=392)
[3]	ModelGrouping (id=393)
[4]	ModelGrouping (id=394)
[5]	ModelGrouping (id=395)
[6]	ModelGrouping (id=396)
[7]	ModelGrouping (id=397)
[8]	null
[9]	null
modCount	8
parentElement	ModelElement (id=351)
size	8

This is what the list looks like:

[, graphics , tabular_graphics_data , output_precision , iterator_servers , iterator_self_scheduling , iterator_static_scheduling , hybrid multi_start pareto_set single_method]

The ModelGrouping we are looking for that holds single_method is shown below. Notice it contains the list of ModelElements:

[7]	ModelGrouping (id=397)
elementData	Object[10] (id=422)
[0]	ModelElement (id=423)
[1]	ModelElement (id=424)
[2]	ModelElement (id=425)
[3]	ModelElement (id=426)
[4]	null
[5]	null
[6]	null
[7]	null
[8]	null
[9]	null
modCount	4
parentContainerReference	ModelContainer (id=378)
selectedElement	null
size	4

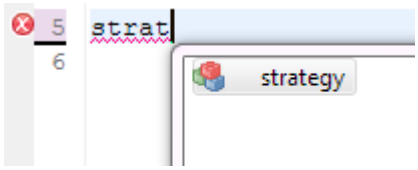
The ModelElement single_method:

[7]	ModelGrouping (id=397)
elementData	Object[10] (id=422)
[0]	ModelElement (id=423)
[1]	ModelElement (id=424)
[2]	ModelElement (id=425)
[3]	ModelElement (id=426)
aliasGroup	55
arrayLengthContainer	null
beforeComment	"" (id=366)
changes	null
defaultInDescription	true
defaultInGroup	false
defaultReal	0.0
defaultString	"" (id=366)
description	" http://www.cs.sandia.gov/dakota/licensing/vot...
elementKeywordLengt	-1
elementKeywordOffse	-1
elementValueOffset	-1
grouping	7
groupName	null
headerInDescription	null
headerInGroup	null
inGUI	true
inlineComment	"" (id=366)
instance_enabled	false
instance_value	null
kind	8
lowerBound	0.0
name	"single_method" (id=438)
numElements	1
parentGroupingRefere	ModelGrouping (id=397)
pausePropChange	false
pointer	ModelContainer (id=439)
prettyName	"Single method strategy" (id=440)
prettySubgroupName	null
required	1
topLevel	false
upperBound	0.0

Section 2.2 AUTOCOMPLETION

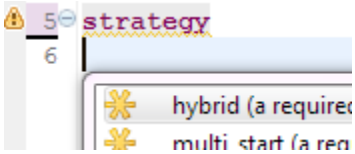
JAGUAR text editor's biggest feature is autocompletion. Autocomplete is triggered by pressing Ctrl-Space. However, there are two modes:

1. **Autocomplete an existing word** (character detected at cursor)



   - The suggestions will only list what is valid to complete the previous word

2. **Autocomplete from scratch**



   - The suggestions will list what is valid at the current position
   -

To understand JAGUAR's autocompletion features, we need to understand an important point: since the grammar has variable-depth, autocompletion needs to work for even elements listed earlier. For example:

```
method
    dot_bfgs
        optimization_type
```

When autocompleting after *optimization_type*, we could be interested in options at the following depths:

At ***optimization_type***:

```
method
    dot_bfgs
        optimization_type
            minimize
```

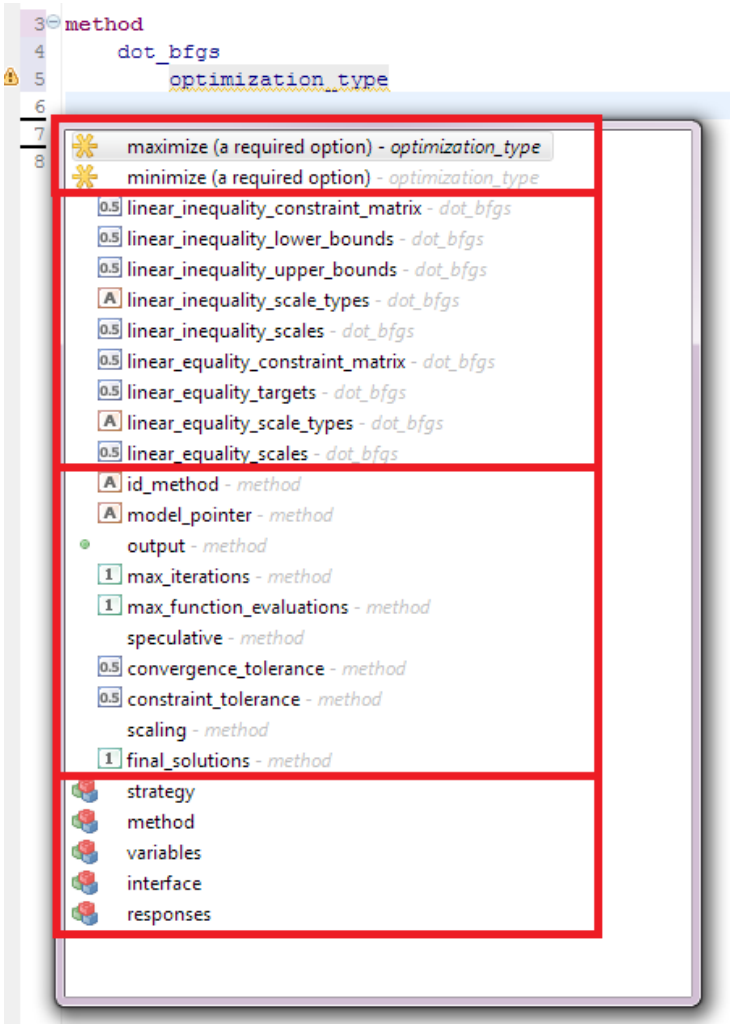
At ***dot_bfgs***:

```
method
    dot_bfgs
        optimization_type
        linear_inequality_lower_bounds
```

At ***method***:

```
method
    dot_bfgs
        optimization_type
    model_pointer = ''
```

As you can see, autocompletion can yield many options. Let's take a look at the actual options available after *optimization_type*:



Four groups of suggestions are presented in order of decreasing locality. Let's take a closer look:

1. **First group** (*optimization_type in trailing gray*)

   The first thing to notice is the trailing keyword in italicized grey ('*optimization_type*') which indicates the originator or parent, and is also highlighted in the text prior. The asterisk on the left indicates one of these required options should be selected (but currently none is selected).

2. **Second group** *(dot_bfgs in trailing gray*)

   These are keywords that are valid and derived from *optimization_type*'s originator, *dot_bfgs.* Notice the type icons on the left, e.g., '1', '0.5', and 'A', indicating that integer, real, and string values respectively are valid.

3. **Third group** (*method in trailing gray*)

   These are keywords that are valid directly in a *method* specification. Notice some have no type icons, representing keywords that do not require an associated value. The keyword *output* has a green dot icon, representing it has derived (child) keywords.

   Since *method* is a top-level section keyword, all remaining section options appear in the fourth group.

4. **Fourth group** (*section*)

   These are the top-level DAKOTA input sections (*strategy, method, variables, interface, responses*) that are applicable at the current context. Since main sections can be inserted anywhere, these are usually available for autocompletion, though DAKOTA syntax may restrict the number of them that can be created.

Section 2.3 DAKREORDER

Dakreorder processes an input deck and essentially returns a formatted version of it, namely expanded abbreviations and cleaned up syntax. Since the formatting changes the original text, dakreorder is only used on the inputdeck when going from text to UI view.

The dakreorder execution can be found in Importer.java runDakotaReorder() method and runs provided the pre-generated specsum file (See Section 1.3).

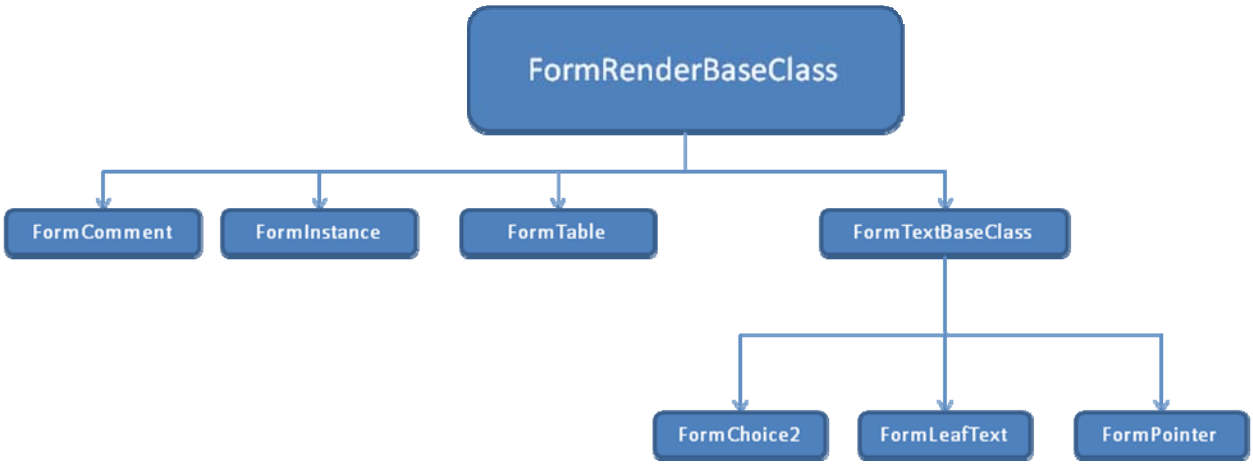
```
dakreord.exe specsum
```

It then accepts standard input, where the input deck is usually passed in. The standard output is the formatted text.

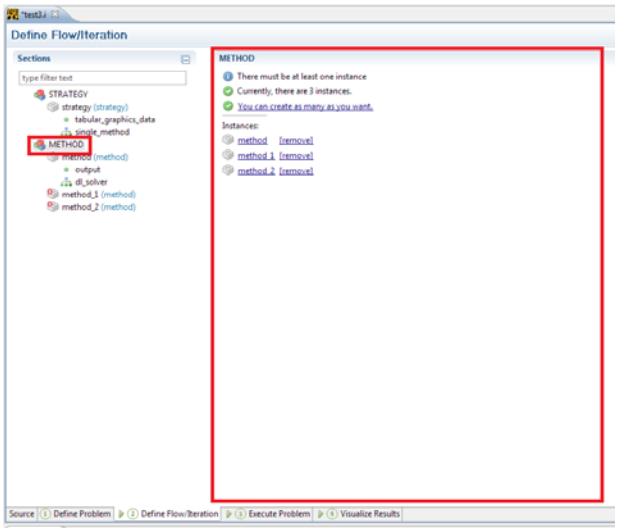
Chapter 3 GRAPHICAL INTERFACE

Section 3.1 BASIC TYPES

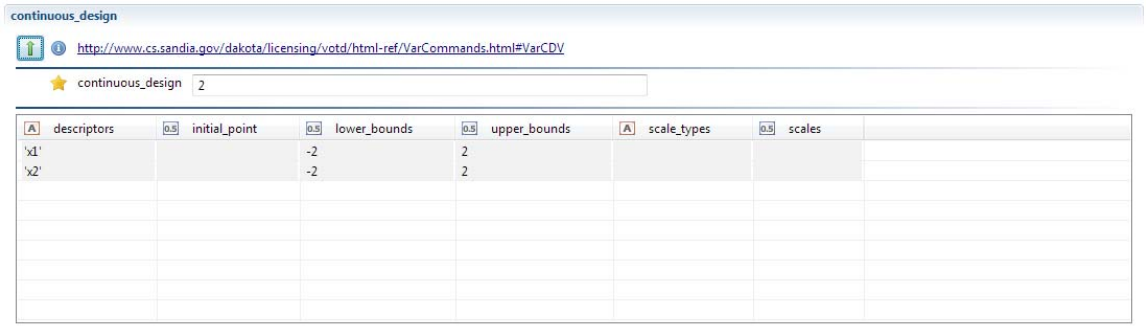
The graphical interface reads off the model and depending on the type of ModelElement, it will generate one of the following widgets:



- **FormComment**: If the element has comments (see 3.2.4)
- **FormInstance**: The selected section and the instances listed:

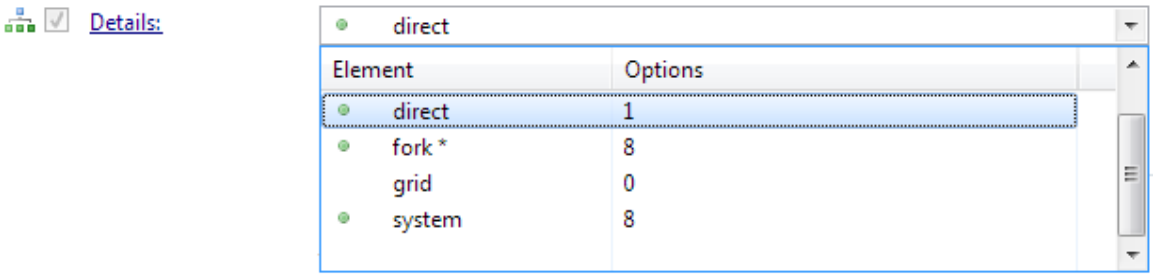


- **FormTable:** Whenever a field in section *Variables* is detected, its subelements will be drawn as columns in a table:



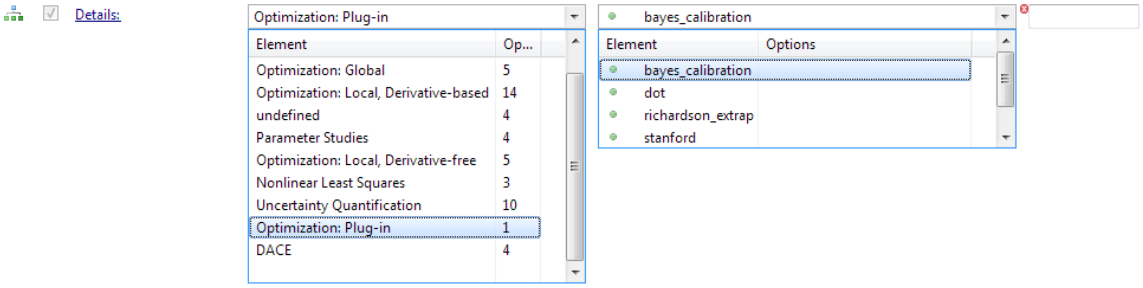
  - You can think of the keyword and values as transposed.

- **FormChoice2:** An updated dropdown using the opensource Nebula project's TableCombo (http://www.eclipse.org/nebula/widgets/tablecombo/tablecombo.php) to render images and multi-column dropdowns.

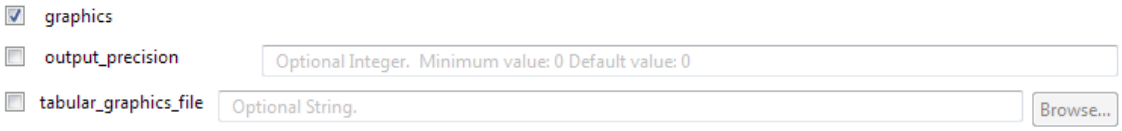


To note, there are certain keywords that require more advanced dropdown rendering:

Notice there's a notion of category (the left dropdown), which is defined by the grammar definition (See Section 1.4). Also there are a few definitions that require a value, as shown as a text field on the right.

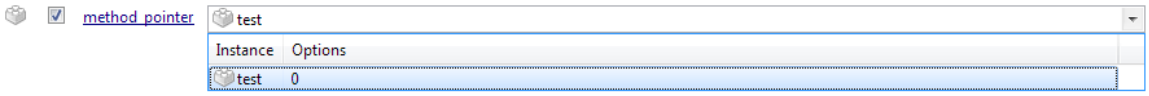


- **FormLeafText** This is the most basic element where the keyword is displayed. There are three main types:

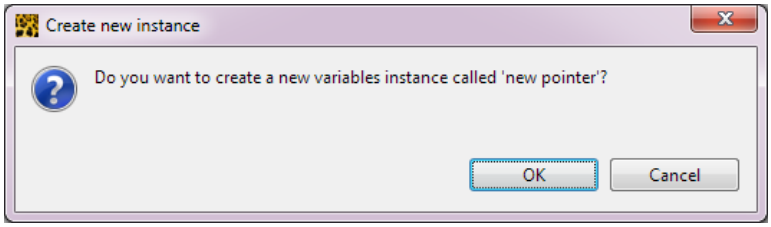


  - **No value:** the element has no value.
  - **Has value**: the element has a value associated with it, either integer, real or string.
  - **File value:** this element has a string value, but also allows the GUI to include a "Browse" button to easily select a file on the user's system. This is created by detecting the string "_file" at the end of the element name.

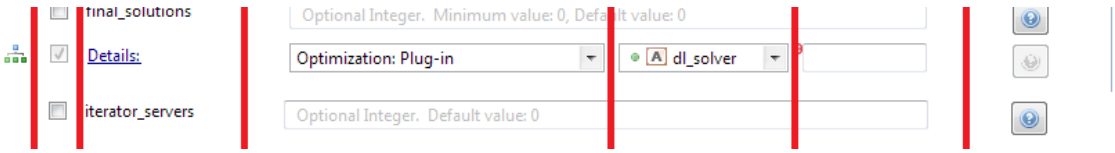
- **FormPointer**: In essence, this is a special type of FormLeafText since it is just a string valued element. This is created by detecting the string "_pointer" at the end of the element name.



The hyperlink allows the user to go directly to the pointer. If it is not already defined, the user is prompted to create it:



It is also noteworthy to mention the forms are laid out using TableWrapLayout since it is able to maximize the width of the UI. However, each element must call setLayoutData() (otherwise unintuitive errors will arise!), namely setting how many columns each element should span. By default each element takes only 1 of the 7 total columns. Below is a picture showing how the 7 columns are set:



Notice *dl_solver* uses each column for a UI widget. However, *iterator_servers*'s textfield covers 3 columns and thus aligns correctly.

It is also important to mention the TableWrapData has two modes TableWrapData.FILL and TableWrapData.FILL_GRAB, the latter grabs maximum horizontal spacing.

Section 3.2 GUI TOOLBAR

The graphical editor has a toolbar on the top right for users to quickly toggle/execute commands.

3.2.1 DAKOTA CHECK

A quick way to invoke DAKOTA to check the input deck for errors. This is equivalent to running 'Check' via the Execute Problem tab.

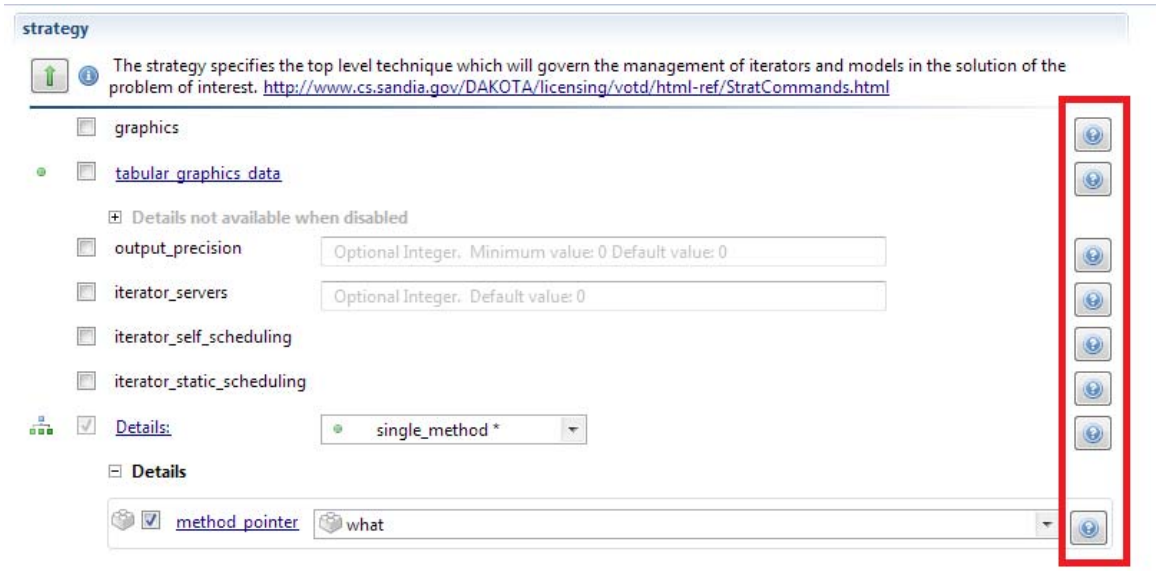
3.2.2 PRETTY NAME

Each element in the grammar has a 'name' and a 'pretty name'. The 'name' is the actual text used in the input deck, whereas the pretty name is a user friendly description of the element.

algebraic_mappings	=	Algebraic mappings file
analysis_components	=	Additional identifiers for use by the analysis_drivers

3.2.3 ONLINE REFERENCE

Each element usually has a description, often with a hyperlink to the documentation. This is visible in the GUI on the right side when the button is selected:



3.2.4 COMMENTS

In this following example:

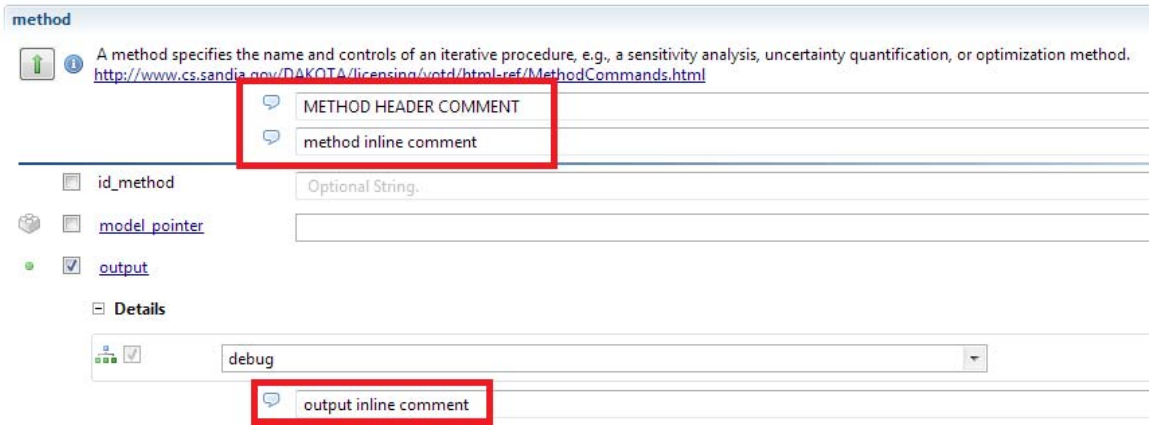
```
## METHOD HEADER COMMENT ##
method #method inline comment
    output #output inline comment
```

It should be noted there are 2 types of comments the above example:

1. Inline (on the same line as a keyword)
2. Before (standalone comments, multiple counts allowed)

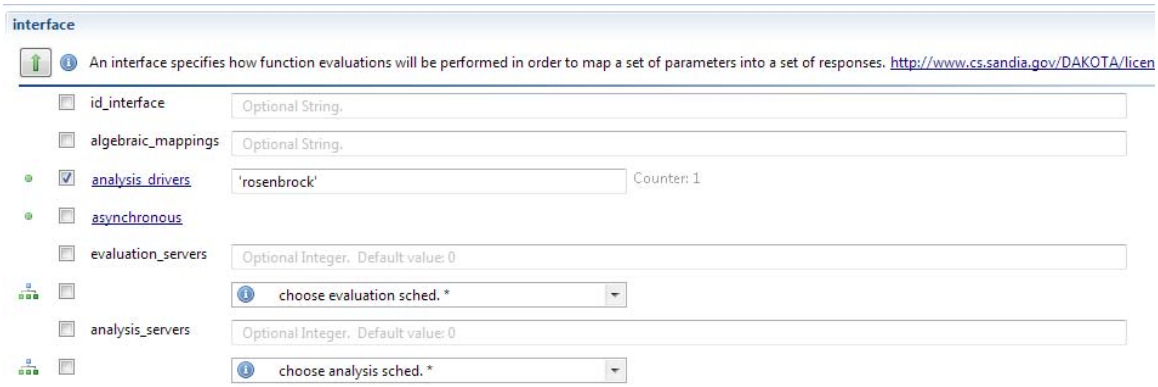
To note, there is no trailing comment support at the moment.

In the GUI, the comments are rendered appropriately when the button is selected:

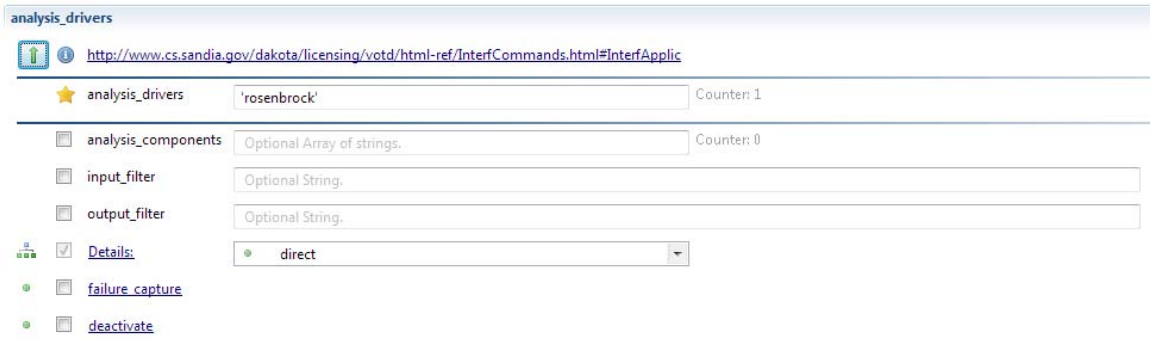


3.2.5 PUSH UP ELEMENTS

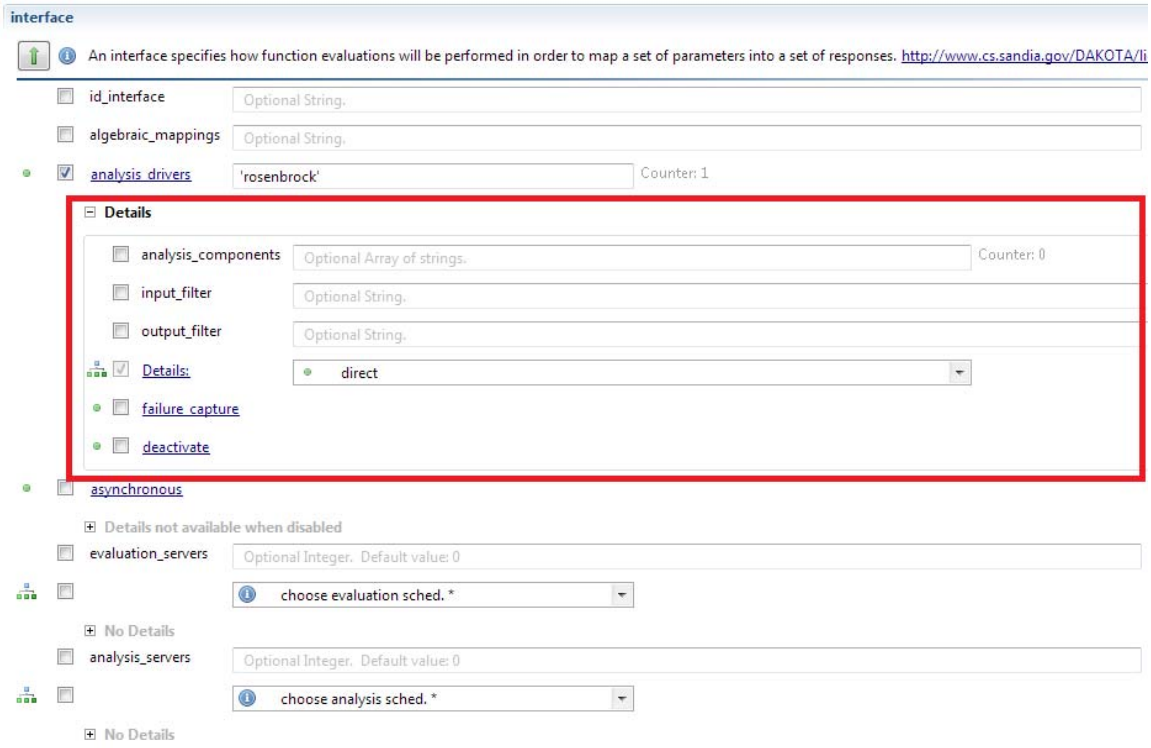
The JAGUAR GUI is simplified by pushup elements. These elements can be displayed in the current pane instead diving deeper in the tree to see them. This helps show the larger context in one view. Pushup elements can be collapsed by clicking on '*Details'*. The images below show JAGUAR with push-up elements on (default) and off, respectively.



In order to edit details within *analysis_drives*, we would normally need to go in and edit the fields:



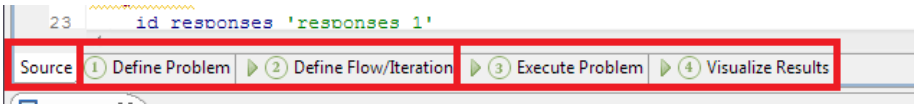
However with pushup elements, we get a hybrid view of subelements on the same page. Notice the new section of *analysis_drivers* is now visible in the *interface* view:



By enabling pushup elements, elements one level deep will be pushed up into the current view. To go into the subelements (level 2), the user will have to click the hyperlinks on the pushed up element.

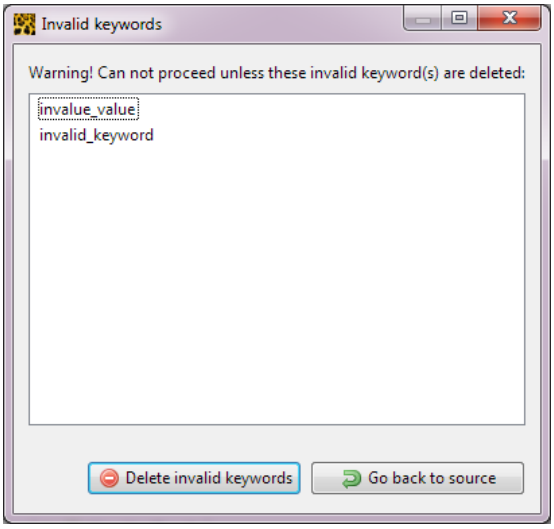
Chapter 4 UI TRANSITION

In the text editor, there are 3 main panels to manipulate the input deck:



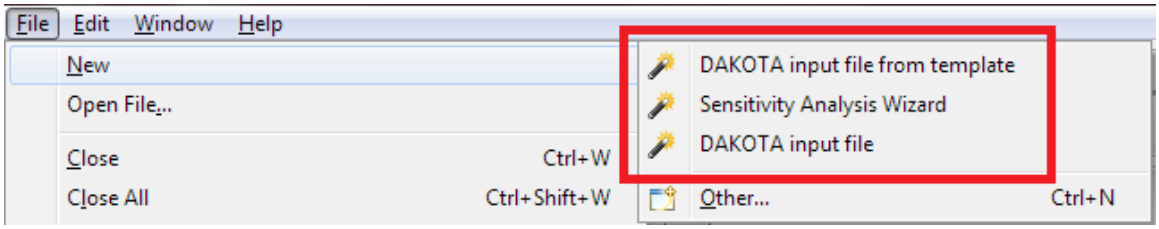
1. Source – the raw text input deck
2. GUI - graphical representation of input deck
3. Execute/Visualize – post processing of input deck

When JAGUAR is switching from text to GUI view, the source text must be parsed, validated and inserted into the model before generating the graphical view. As such, any erroneous keywords must be removed before rendering, otherwise they will be lost forever. JAGUAR prevents accidental deletion by prompting the user:



The user has a choice to fix the keyword, or to continue with the invalid keywords removed.

Chapter 5 WIZARDS

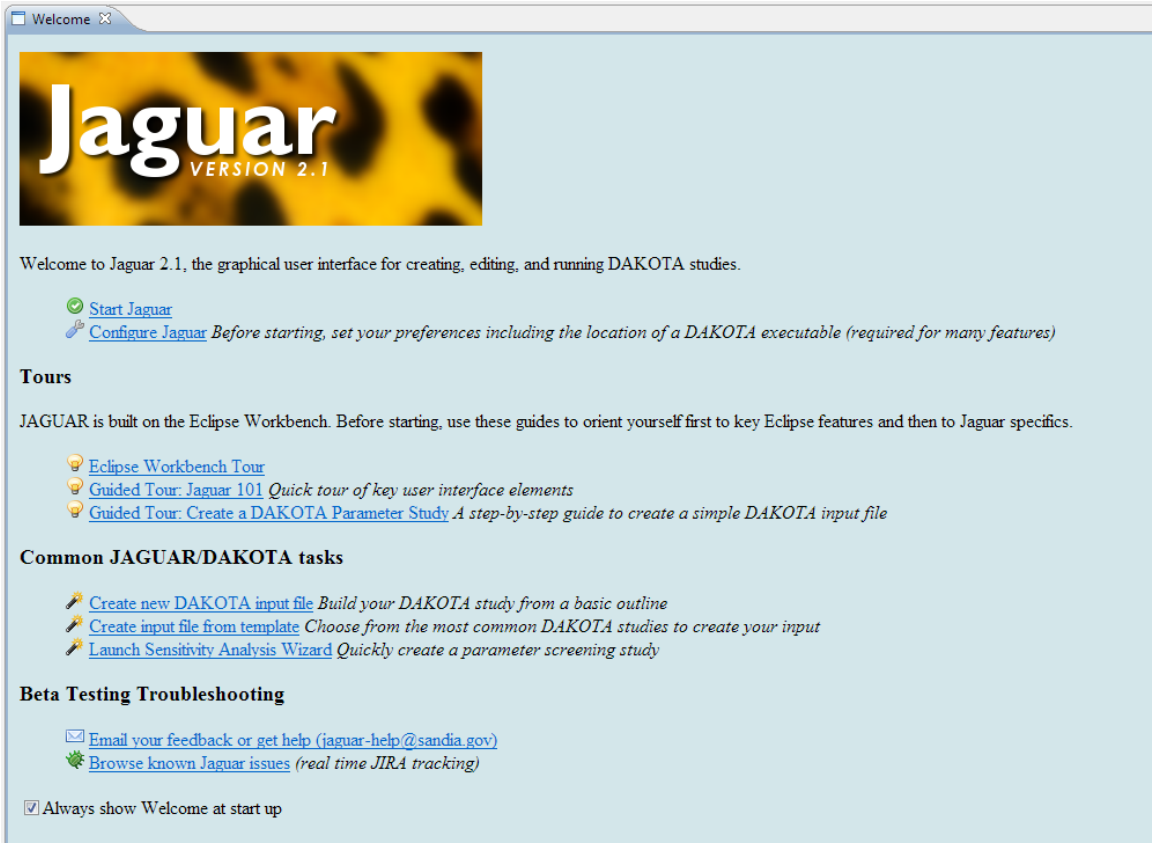


The wizards can be found in */gov.sandia.dart.jaguar/src/gov/sandia/dart/jaguar/wizard*

Chapter 6 WELCOME SCREEN

This is the screen that users see by default and is a quick way to introduce the options.

Relevant files can be found at: */gov.sandia.dart.jaguar/src/gov/sandia/dart/jaguar/intro*



The welcome screen *(main.html)* is an html page with special tags that map to actions:

```
<a href=http://org.eclipse.ui.intro/runAction?class=gov.sandia.dart.jaguar.intro.NewLHSWizard&amp;pluginId=gov.sandia.dart.jaguar>
```

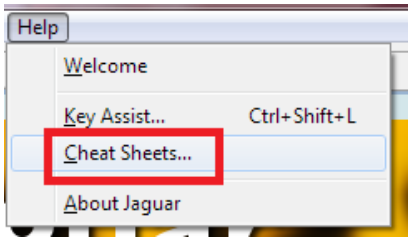
Additionally to note, the checkbox at the bottom “Always show Welcome at start up” is added through *introContent.xml* with this code:

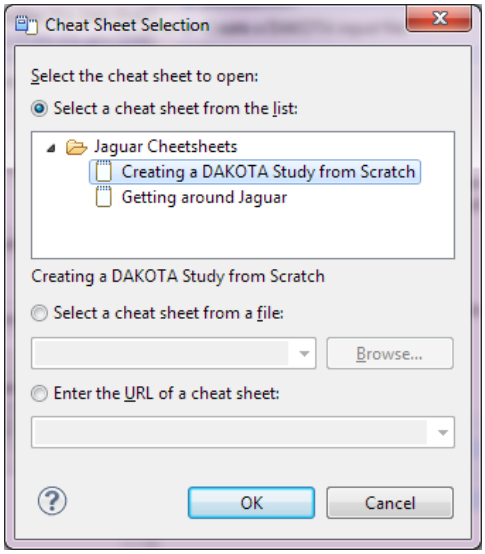
```
<contentProvider id="gov.sandia.jaguar.alwaysdisplaycheckbox"
class="org.eclipse.ui.intro.contentproviders.AlwaysWelcomeCheckbox" >

        <text>AlwaysWelcomeCheckbox will not be displayed in Eclipse pre-3.5 binary....</text>

</contentProvider>
```

Chapter 7 CHEATSHEETS

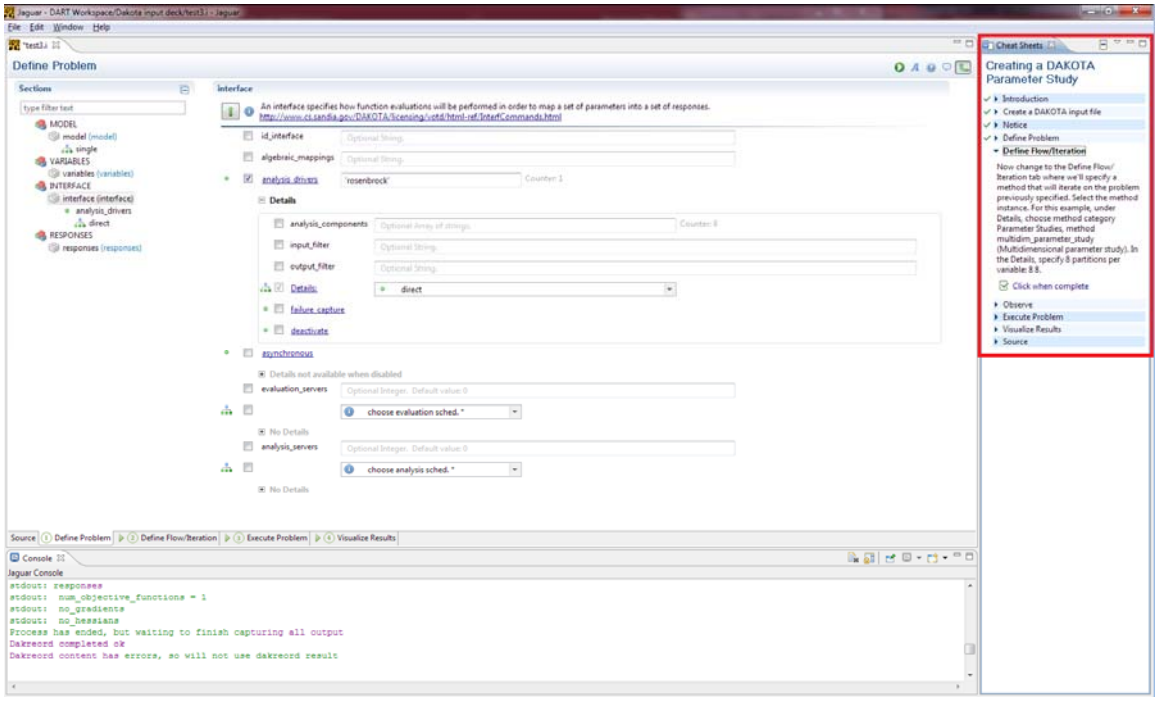
Cheat sheets definitions can be found in */gov.sandia.dart.jaguar/cheatsheet* and enabled in *plugins.xml*

For quick tutorials, Help -> Cheat Sheets





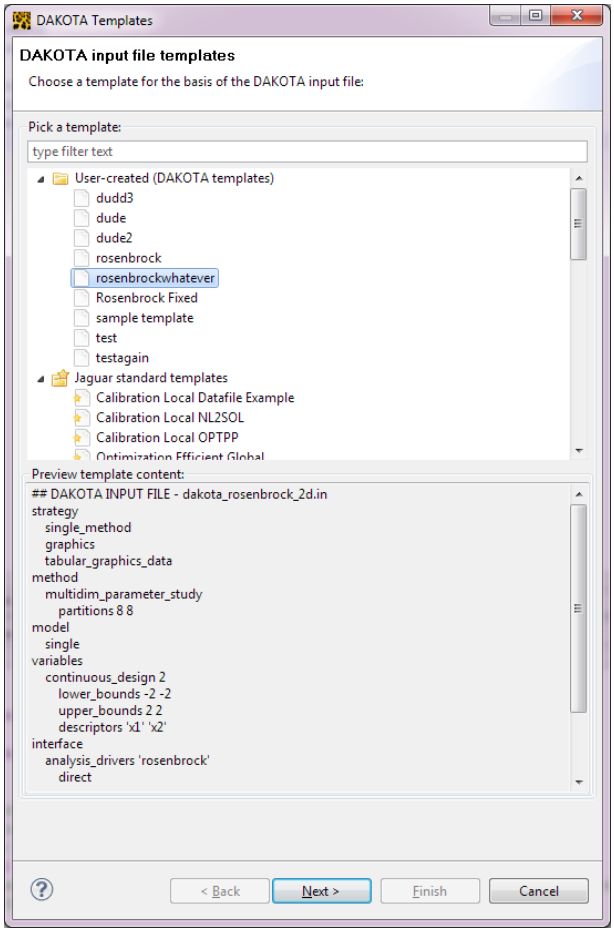
Cheatsheets are basically guided tutorials for the user to follow.



There are plans later on to incorporate dynamic help.

Chapter 8 TEMPLATES

To help users effectively create input decks, DAKOTA includes templates that can be used to quickly create standardized input decks that can later be edited. In addition, user-created templates can also easily be saved and restored.

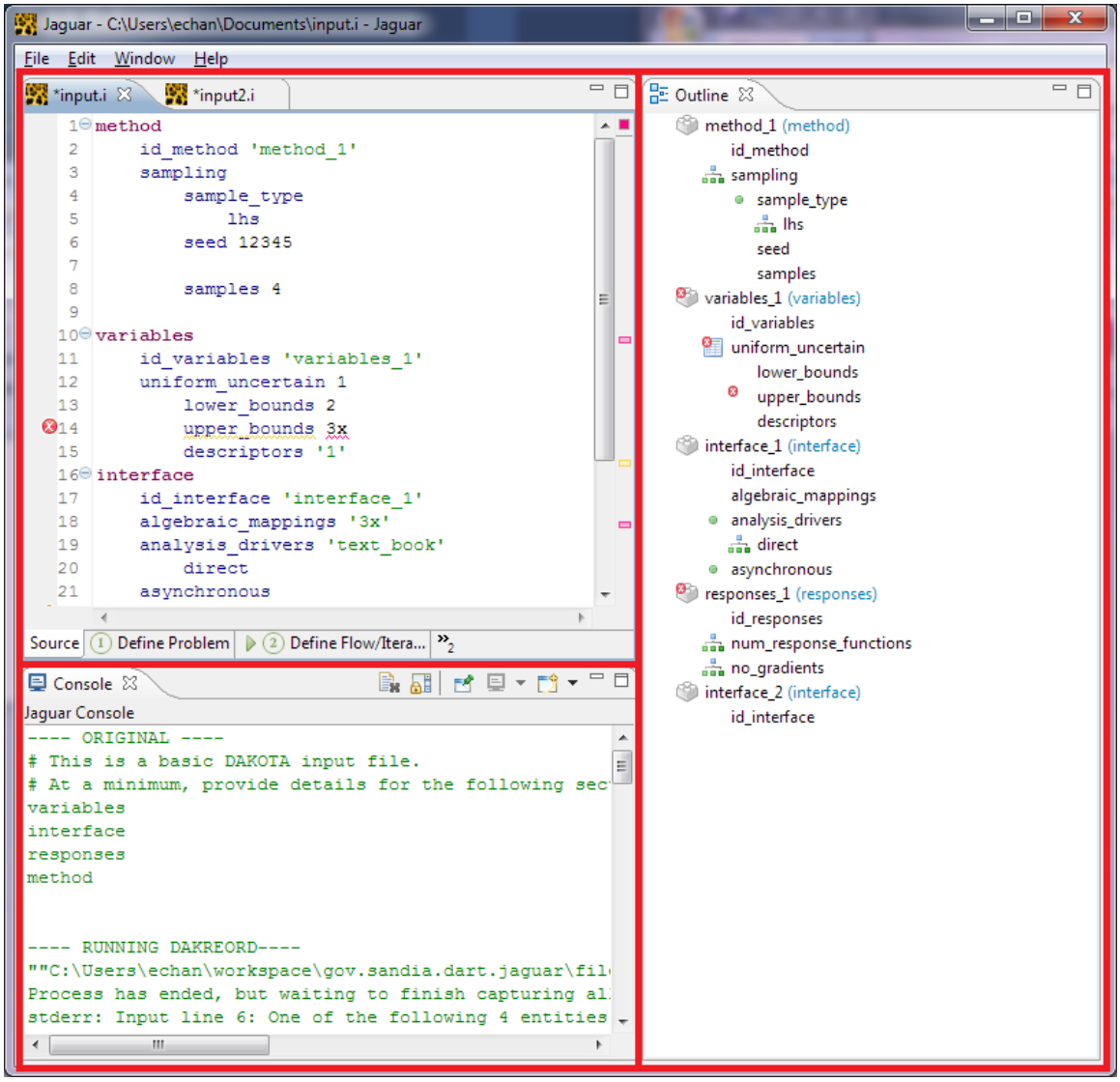


This is the template wizard. At the top half are the available templates, both user-created and JAGUAR's built-in templates. And at the bottom half a preview of the selected file.

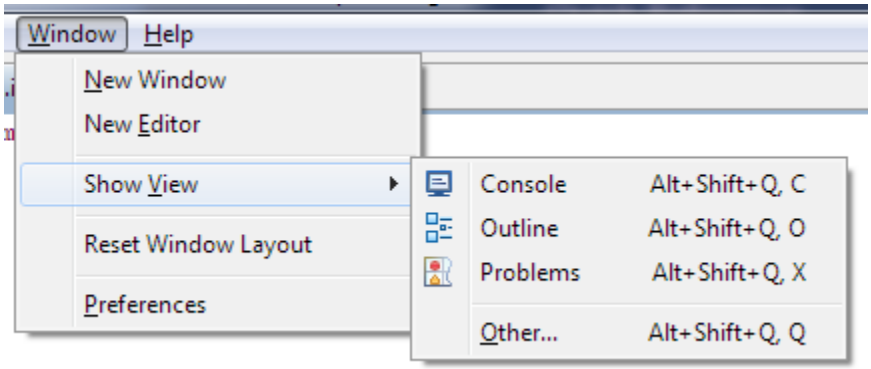
The user templates can be found through PreferenceStore's *template_path*, whereas the JAGUAR built-in templates can be found */gov.sandia.dart.jaguar/files/templates.*

Chapter 9 VIEWS

Views in JAGUAR are single-purpose windows. The default JAGUAR has 3 views, the main editor (top left), Outline View (right) and Console View (bottom left).



Additional Views can be added through Window→Show View.

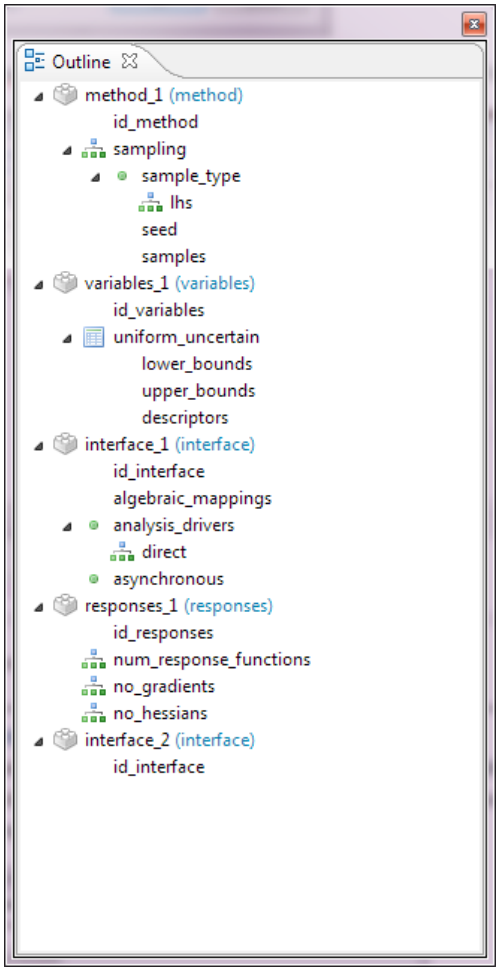


Section 9.1 OUTLINE VIEW

The Outline View strips the input deck leaving only the keywords behind. This aids the user with a view of the input deck without values, comments and textual cosmetics.

Clicking into the tree allows quick reference to the Source View.

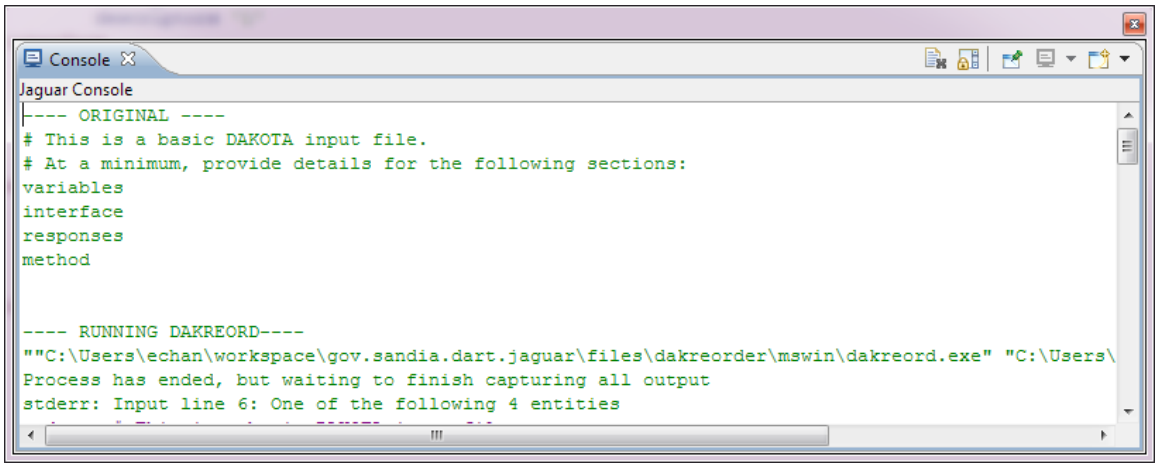
The source code can be found:
/gov.sandia.dart.jaguar/src/gov/sandia/dart/jaguar/editors/partition2/JaguarEditorOutlinePage.java



It shares the same code used in the GUI view's left sidebar, except it is configured to display all leaf nodes and only sections that are defined (Note: GUI view shows all sections (enabled or not) and only nodes with children).

Section 9.2 CONSOLE VIEW

The Console View shows supplemental information on what JAGUAR is running, which is usually dakreorder. This view is not required, and is most helpful when determining problems.

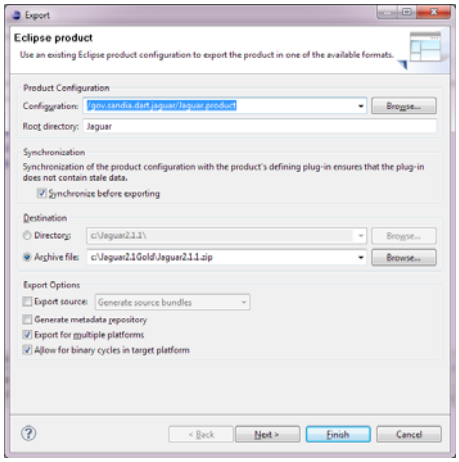


Chapter 10 CREATING BINARIES

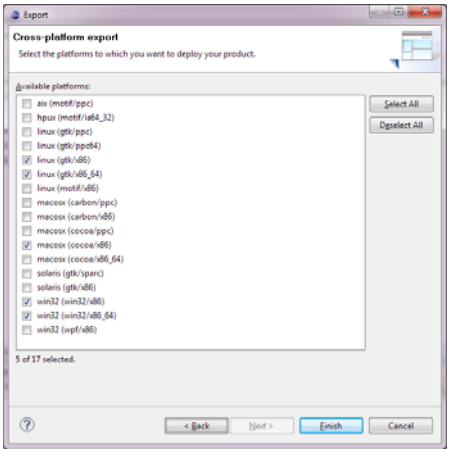
Section 10.1 EXPORTING IN ECLIPSE

JAGUAR is able to export to multiple platforms: Windows, Mac OSX and Linux. Assuming all dependencies and fragments are configured correctly in /gov.sandia.dart.jaguar.feature/feature.xml, creating the binaries is as follows:

In /gov.sandia.dart.jaguar/Jaguar.product, under “Exporting”, select “Eclipse Product export wizard”.



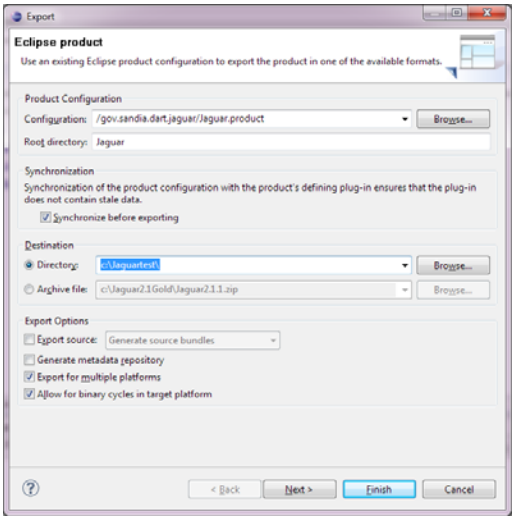
Under Destination, Select “Directory” for creating the actually files/directories, or “Archive file” for a compressed form. For creating binaries, we usually use the “Archive file” option.



Next screen allows the user to select available platforms. JAGUAR 2.1 is tested on linux (gtk/x86), linux (gtk/x86_64), macosx (cocoa/x86), win32 (win32/x86), win32/x86_64).

Section 10.2 CREATING WINDOWS INSTALLER

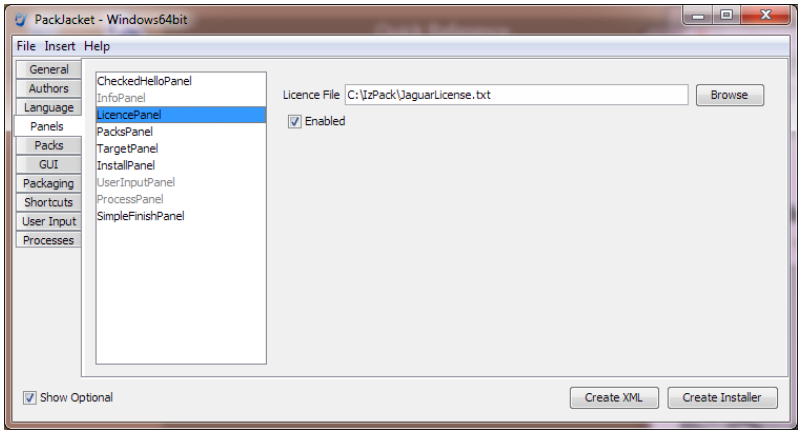
To create Windows executable installer, Under “Destination”, instead of archive, we export to the “Directory”.



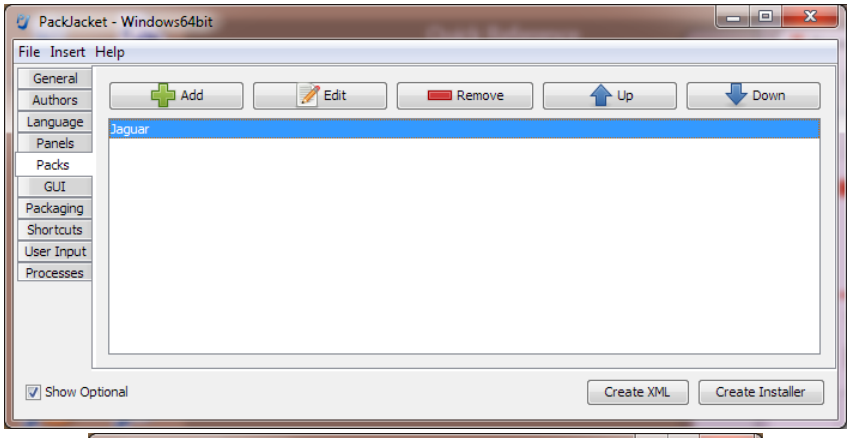
- On the platform page, select only “win32 (win32/x86)”

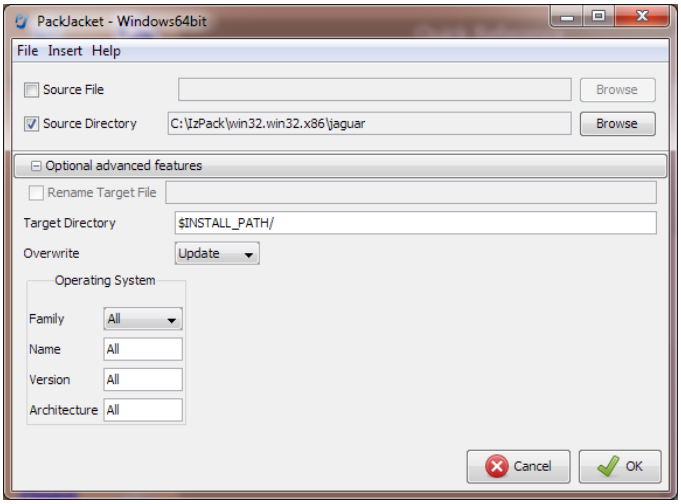
Use PackJacket (http://packjacket.sourceforge.net/) or create an appropriate installation XML for IzPack (http://izpack.org). I’m assuming we’re using PackJacket.

Check the license file is in the correct place.

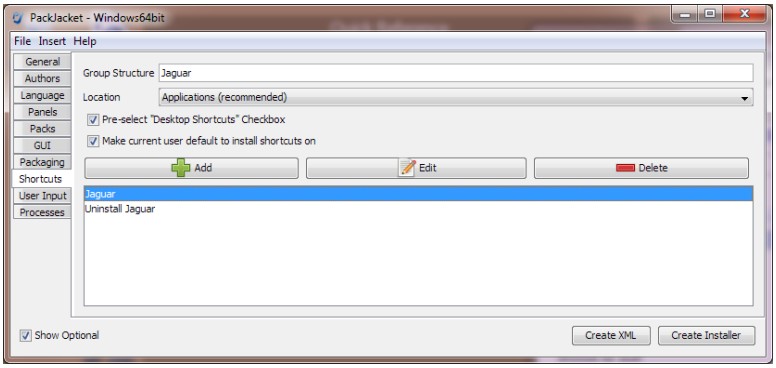


Check the Packs, especially check the “source Directory” matches where the exported JAGUAR folder is located, because that’s where PackJacket is going to look to create the installer binary.

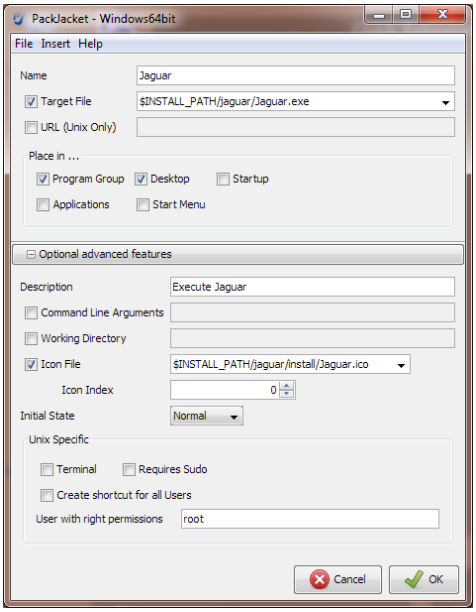




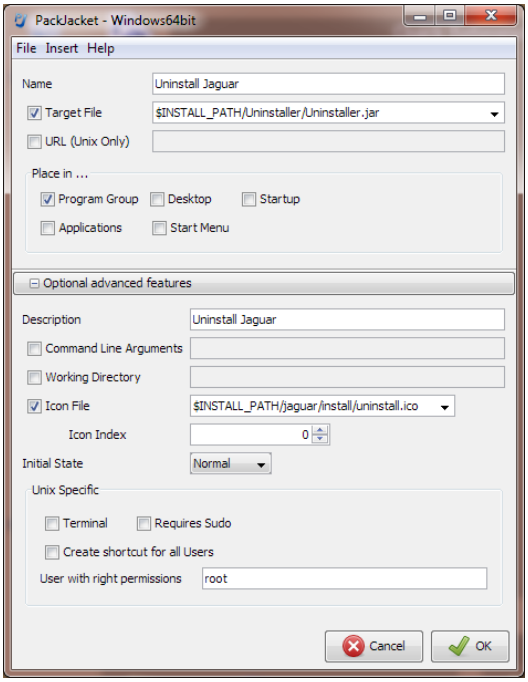
Check that the Shortcut is set correctly:



Check icon and icon file path (make sure the paths after $INSTALL_PATH match the “Root directory” you defined in the Export Wizard). Check “Place in” Program Group and Desktop only.

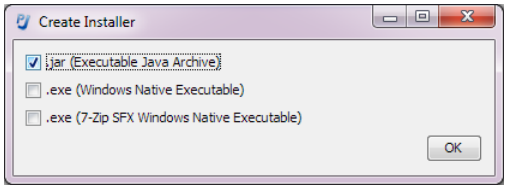


Check icon and icon file path (make sure the paths after $INSTALL_PATH match the "Root directory" you defined in the Export Wizard). Check "Place in" Program Group only.



After all the settings are correctly configured, "Create Installer" and:

- For 32-bit, create .jar (will use izpack2exe to create 32-bit installer)
- For 64-bit, create .exe (creates a 64-bit installer if created in 64-bit environment)



For 32-bit, start a command prompt and go to \utils\wrappers\izpack2exe Sample execution:

```
"izpack2exe.exe --file=c:\IzPack\Jaguar2.1.jar --output=c:\IzPack\Jaguar.exe"
```

Section 10.3 Creating Mac Installer

Export to Directory from Eclipse, and use DMG Canvas (www.araelium.com/dmgcanvas) to create a Mac disk image (.dmg file) allowing users to easily install JAGUAR to their Applications folder. Make sure JAGUAR's reference is to the exported directory from Eclipse.

Section 10.4 Releasing

Upload all the .zip archived files, .dmg installer and .exe installers to a new folder under \\snl\Collaborative\DAKOTA\DART_GUI\CurrentPrototype

Distribution

1	MS0899	Technical Library	9536 (electronic copy)


Sandia National Laboratories