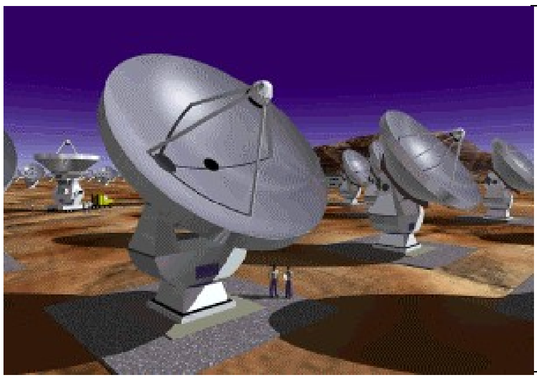


Atacama Large Millimeter

KGB-DOC-01/09

Revision: 1.7

2006-11-07

User's manual

Mihael Kadunc

Object Explorer

User's manual

Mihael Kadunc

Josef Stefan Institute, Ljubljana

Gašper Tkačik

Josef Stefan Institute, Ljubljana

Matej Šekoranja

Josef Stefan Institute, Ljubljana

Keywords:	
Author Signature:	Date: 6.7.2001
Approved by: Institute:	Signature: Date:
Released by: Institute:	Signature: Date:

Change Record

REVISION	DATE	AUTHOR	SECTIONS/PAGES AFFECTED
	REMARKS		
1.0	2001-07-06	Mihael Kadunc	All
	Created		
1.1	2001-09-09	Mihael Kadunc	All
	Updated according to comments by Gianluca Chiozzi, ESO		
1.2	2001-09-14	Gašper Tkačik	Section 6
	Added an explanation of Engine functionality.		
1.3	2001-09-24	Mihael Kadunc	Section 6
	Updated package names to si.ijs.acs.*		
1.4	2002-01-05	Mihael Kadunc	Table of contents
	Added table of contents according to ACS Remaining List doc-12		
1.5	2002-02-13	Mihael Kadunc	Sections 3-5
	Updated for the new version of Object Explorer (1.1)		
1.6.	2003-11-11	Matej Šekoranja	Section 6.
	Updated BACI plug section to match ACS 3.0 MACI IDL, added `objexp.pool_timeout`.		
1.7	2006-11-07	Matej Šekoranja	Section 6.3, section 7.3 inserted.
	Supported types update. Value conversion section added.		

Table of Contents

1 Introduction

Object Explorer (OE) is a generic tool used for low-level inspection of objects in a software environment like BACI or AbeansR2. It can be used as a debugging or testing tool by the developers and maintainers of a control system.

Basically, OE is split into two parts:

- the **GUI part** (si.ijs.acs.objectexplorer package), which is independent of the inspected control system
- the **Engine**, which is different for each control system and communicates with object servers. Both parts communicate through interfaces defined in the si.ijs.acs.objectexplorer.engine package. The main part of the engine is its *RemoteAccess* class, which provides the connection to the inspected system.

2 Application overview

User connects to a remote system by clicking the *Connect* option in *File* menu and selecting the type of remote access he would like to use. Objects are shown in the tree below the button. By selecting an object, its operations and attributes are shown in the lists on the right side of the window. User invokes operations and inspects attributes by double-clicking on them in the lists. Operations that require parameters will show a *Parameter window,* and the user is asked to insert parameter values into appropriate fields. Results of these actions will be reported to the *Result text area* or to the *RemoteResponse window*.

There are two different windows in the OE application – *OE main window*, used for connecting to objects, invoking their operations and inspecting attributes, and *RemoteResponse window*, where responses of the invocations are shown.

There is also the *Error dialog*, which appears every time an exception occurs, *Parameters dialog* which is used for acquiring parameters for method invocations, and *RemoteAccessDestroy dialog* which appears when OE is disconnecting from the control system and monitors the status of the disconnecting process.

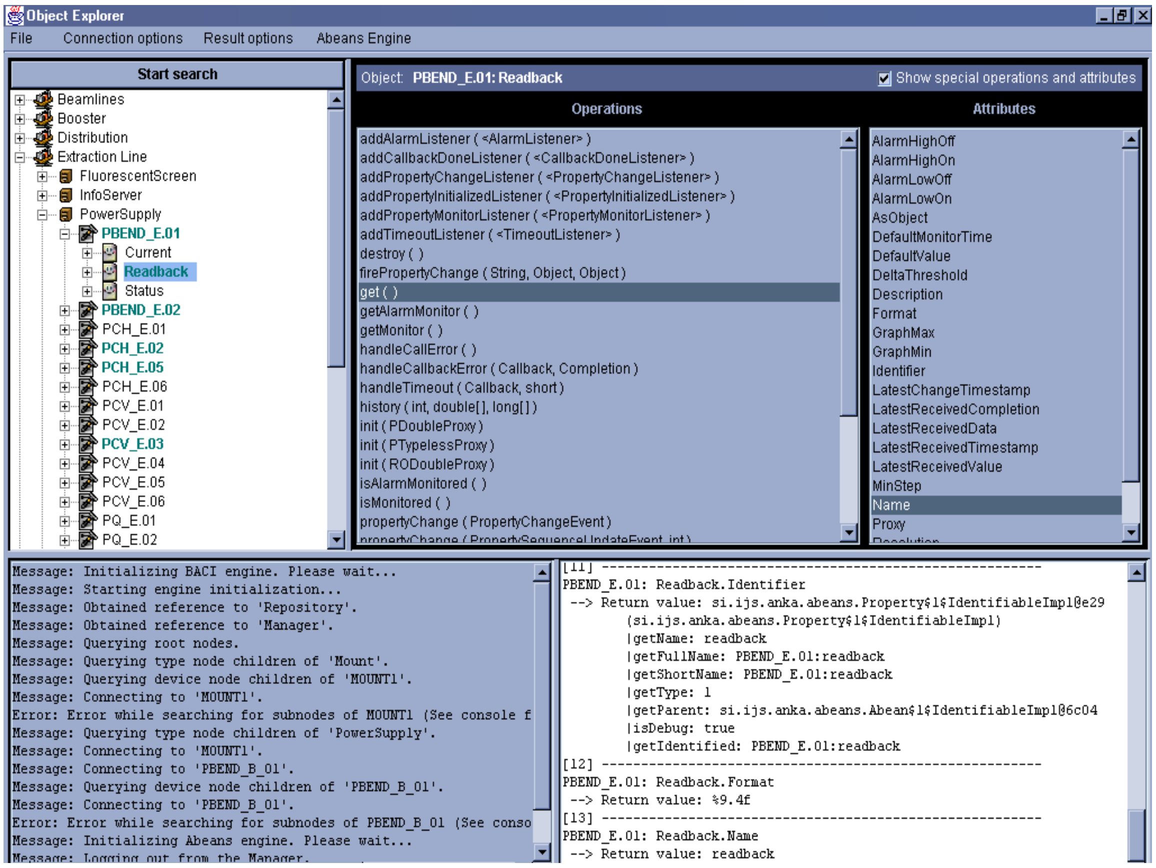
3 Running Object Explorer

Object explorer can be started in various ways in the ACS environment:

- Commnad line

- Command center
- Web start
- Java class

4 Main window



```
addAlarmListener ( <AlarmListener> )
addCallbackDoneListener ( <CallbackDoneListener> )
addPropertyChangeListener ( <PropertyChangeListener> )
addPropertyInitializedListener ( <PropertyInitializedListener> )
addPropertyMonitorListener ( <PropertyMonitorListener> )
addTimeoutListener ( <TimeoutListener> )
destroy ( )
firePropertyChange ( String, Object, Object )
get ( )
getAlarmMonitor ( )
getMonitor ( )
handleCallError ( )
handleCallbackError ( Callback, Completion )
handleTimeout ( Callback, short )
history ( int, double[], long[] )
init ( PDoubleProxy )
init ( PTypelessProxy )
init ( RODoubleProxy )
isAlarmMonitored ( )
isMonitored ( )
propertyChange ( PropertyChangeEvent )
```

```
AlarmHighOff
AlarmHighOn
AlarmLowOff
AlarmLowOn
AsObject
DefaultMonitorTime
DefaultValue
DeltaThreshold
Description
Format
GraphMax
GraphMin
Identifier
LatestChangeTimestamp
LatestReceivedCompletion
LatestReceivedData
LatestReceivedTimestamp
LatestReceivedValue
MinStep
Name
Proxy
```

```
Message: Initializing BACI engine. Please wait...
Message: Starting engine initialization...
Message: Obtained reference to 'Repository'.
Message: Obtained reference to 'Manager'.
Message: Querying root nodes.
Message: Querying type node children of 'Mount'.
Message: Querying device node children of 'MOUNT1'.
Message: Connecting to 'MOUNT1'.
Error: Error while searching for subnodes of MOUNT1 (See console f
Message: Querying type node children of 'PowerSupply'.
Message: Connecting to 'MOUNT1'.
Message: Connecting to 'PBEND_B_01'.
Message: Querying device node children of 'PBEND_B_01'.
Message: Connecting to 'PBEND_B_01'.
Error: Error while searching for subnodes of PBEND_B_01 (See conso
Message: Initializing Abeans engine. Please wait...
Message: Logging out from the Manager.
```

```
[11] ----------------------------------------------------------
PBEND_E.01: Readback.Identifier
 --> Return value: si.ijs.anka.abeans.Property$1$IdentifiableImpl@e29
        (si.ijs.anka.abeans.Property$1$IdentifiableImpl)
        |getName: readback
        |getFullName: PBEND_E.01:readback
        |getShortName: PBEND_E.01:readback
        |getType: 1
        |getParent: si.ijs.anka.abeans.Abean$1$IdentifiableImpl@6c04
        |isDebug: true
        |getIdentified: PBEND_E.01:readback
[12] ----------------------------------------------------------
PBEND_E.01: Readback.Format
 --> Return value: %9.4f
[13] ----------------------------------------------------------
PBEND_E.01: Readback.Name
 --> Return value: readback
```

OE window is divided into four sections: Object tree, lists of object's members, message text area and results text area. There are also the menus at the top of the window.

4.1 Menus

4.1.1 File

File menu provides a *connect* menu item which is used to select the type of the remote access and connect to remote system (Currently supported options are *BACI* and *AbeansR2*) , and *exit* option, which closes the window and exits the program in the same way as if the application was exited by closing the main window.

4.1.2 View

In the *View* menu you can choose the following:

- **Expand result data**: If a result of an action (Operation or Attribute invocation) is not a primitive type (or a wrapper object of one) and this option is checked, OE tries to find all the members of this object recursively and prints a hierarchical representation of the object to the result text area. Otherwise only object's toString() representation is printed. This option is not checked by default.

- **Invocation response in a separate window**: Some invocations (e.g. monitors) send responses continuously, and in order to make the responses clearer, invocations that produce more than one response are filtered out and shown in a separate RemoteResponse window. If this option is not checked, all invocation output goes to the result text area. Option is checked by default.

- **Debug to console**: If this option is checked (default), all debug from OE GUI and engine is printed to the console. Debug is ignored otherwise.

4.1.3 Engine menu

appears when OE is connected to the engine. It is engine-specific, allowing user to configure engine and control its behavior.

4.2 OE Tree

4.2.1 *Search* Button

Restarts the remote session: connects to the selected engine by creating and initializing a *RemoteAccess* class, and searches for the root nodes in the system (domains or other object groups). If another RemoteAccess is already present, OE destroys it allowing it to disconnect from the remote server, before starting a new session. All RemoteResponse windows are closed upon disconnection. RemoteAccessDestroy dialog is shown.

4.2.2 Nodes

Nodes in the tree represent object groups, objects, their properties and invocations. When a node is expanded for the first time, the engine searches for its subnodes. After that nodes cannot be removed from the tree and they can not be refreshed. In order to refresh the whole tree properly, user should press the Start Search button again.

- Objects and properties are *introspectable* nodes – when user clicks on one of them, its operations and attributes are displayed. They also support connecting and disconnecting: User can connect to an object either by right-clicking on the node and selecting *connect* on the popup menu, or by selecting the node in the tree. Connected nodes are displayed with a different color and in bold font. To disconnect an object, right-click on the node that represents it and select *disconnect* on the popup menu. Even if an object is disconnected, its node still exists in the tree. Introspectable nodes cannot be removed from the tree after they have been loaded.

- Invocations are *simple introspectable* nodes – they have operations and attributes, but they cannot be connected to or disconnected. They can, however, be removed from the tree, as they represent only temporary objects. There is no special support for removing invocations in the OE GUI part. Invocations are removed by the engine when the object represented by them is destroyed or they are not needed by the engine anymore (usually engine provides a method to destroy an invocation). Invocations report remote responses which are printed in the remote response window.

- Other nodes in the tree are used as containers for the object nodes and are not controllable in any way.

4.2.3 Connect popup menu

This popup menu appears when user right-clicks on an introspectable node. It shows the object's name and allows user to connect or disconnect the object.

4.3 Operations and Attributes lists

The operations and attributes lists show the selected object's operations (methods) and attributes (fields). When user selects an introspectable object in the tree, its members are found and listed in the two lists. If a node is clicked while the engine searches for the members of another node, the last selection is ignored and members of the node clicked first are displayed.

A label above the lists displays the selected object's name. Next to the label there is a check box which toggles the display of special operations and attributes.

4.3.1 Operations

Operations (methods) are displayed in the left list, represented by their name and parameters types in parentheses. Parameters that are provided by the engine and do not have to be entered by the user are enclosed in <> signs. Array types are represented with a set of [] for each dimension of the array.

To invoke an operation, double-click on it in the list. If no user entry is required, it will be invoked immediately, otherwise a Parameters dialog will appear and operation will be invoked after the user clicks the Invoke button. Operation results (return values) are printed into the Result text area, or into the RemoteResponse window if more than one response is received.

Invocations are objects that are temporarily created when user invokes an operation and are capable of reporting multiple responses. A node representing invocation is displayed in the OE tree as a subnode of the object whose operation caused it. They are removed from the tree when they are no longer needed, i.e. when engine knows they would not produce responses any more.

4.3.2 Attributes

Attributes (fields) are displayed in the right list, represented by the name. Double-click retrieves the attribute value and prints it into the Result text area. Currently, attributes can only be read, setting their values is not implemented yet. Usually objects will have methods used to set their attributes.

4.4 Result text area

Result text area is displayed in the lower-right corner of the OE main window. It contains results from invoked operations and inspected attributes. A result report consists of a serial number, a string representing the object followed by the operation/attribute name, and return value. Auxiliary return values are parameters, passed to the method either by the user or the engine.

Return values are displayed by their String representation or a hierarchical list of attributes – as specified by the user in the *View* menu.

Maximum number of lines displayed in the text area is 1000. If the output is longer than that, the text is trimmed at the beginning.

Text in the text area is not editable (may be on some platforms). However, user can copy the text to the clipboard by typing *CONTROL + C*.

By clicking the right mouse button on the text area a popup menu with two options is displayed:

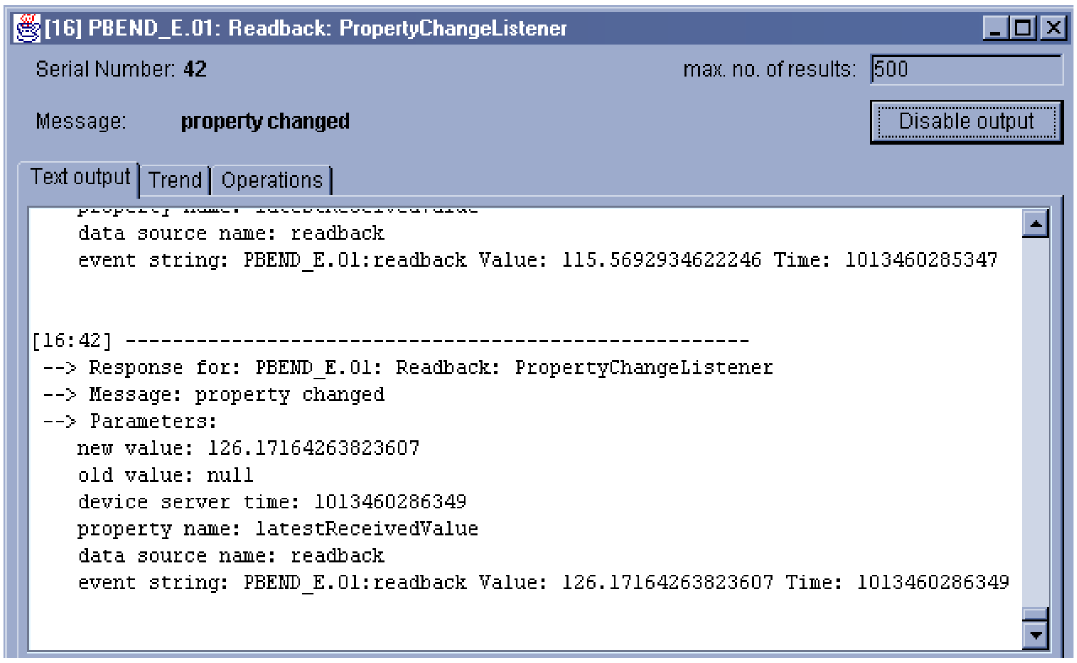
- *Save –* the contents of the text area are saved to a file specified by the user.
- *Expand to dialog* – the text area is removed from the main window and displayed in a separate dialog window. It can be placed into the OE window again by clicking *Dock* in the popup menu or closing the dialog window

4.5 Message text area

Message text area displays messages reported by the GUI or OE engine. When an error occurs, a message is also printed here. Maximum number of lines printed is 1000.

The popup menu on this text area is the same as on the Result text area.

5 RemoteResponse window



Serial Number: **42**
max. no. of results: 500
Message: **property changed**

```
    data source name: readback
    event string: PBEND_E.01:readback Value: 115.5692934622246 Time: 1013460285347


[16:42] -----------------------------------------------------
 --> Response for: PBEND_E.01: Readback: PropertyChangeListener
 --> Message: property changed
 --> Parameters:
    new value: 126.17164263823607
    old value: null
    device server time: 1013460286349
    property name: latestReceivedValue
    data source name: readback
    event string: PBEND_E.01:readback Value: 126.17164263823607 Time: 1013460286349
```

RemoteResponse window shows results of a single invocation. The title bar displays its name and serial number. Two labels display the serial number and the message of the last response received. User can specify the number of responses shown by the window in the *max. no. of results* text field (to prevent excessive memory consumption). To change this number, click on the text field and write a number (must be between 1 and 10^9). Then press enter to update the value. If an invalid number is typed the old value is restored. User can disable all graphical output order to free CPU time by clicking the Disable output button.

User can not close the RemoteResponse window while the invocation is still active (i.e. not destroyed by the engine). See engine specifications (section 6 of this document) to find out how to destroy invocations.

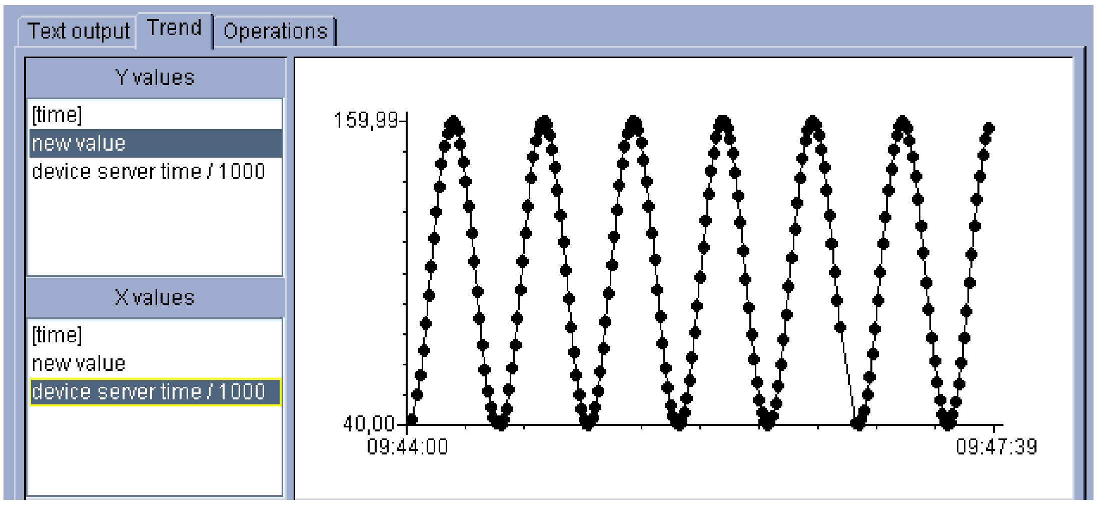
To display responses and manipulate the invocation three panel are available in the window:

5.1.1 Text output panel

(displayed in the above picture) shows the latest remote responses. Each response starts with its invocation name, serial number and message. Then the response data is enumerated. Old responses are deleted from the text area (according to the max. no. of results specified by the user). When the window is minimized, all responses are buffered and printed to the text area only upon restoring the window.

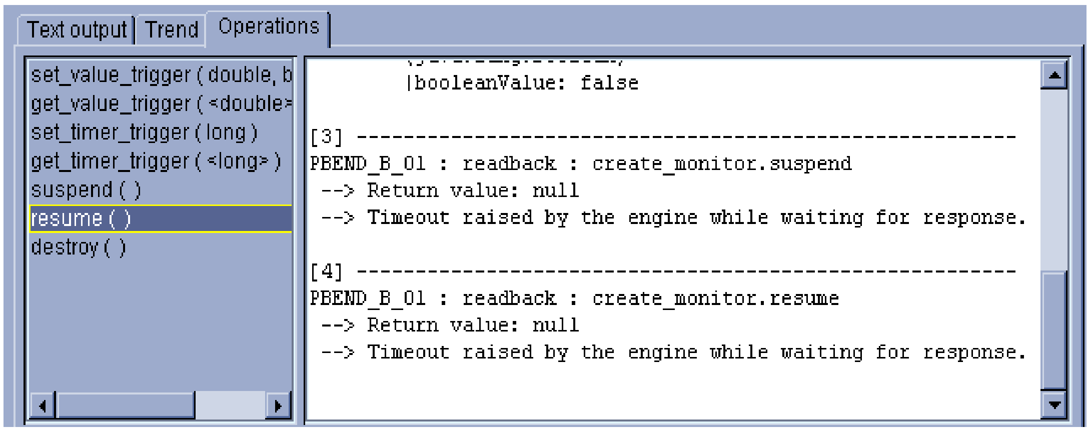
Popup menu on the text area offers *Save* and *Expand result data* options, which allow user to save the text in the area to a file and to expand the data returned by responses.

5.1.2 Trend panel



Trend panel shows the receieved data in a trend graph. User can select the desired Y and X value shown from the lists on the left. Values that can be chosen are [time], which is the system time at the arrival of response to the window, and other numerical data contained in the remote response. Number of points shown on the trend is specified by the *max. no. of results* text field. Scale of the trend chart is determined by the values as they arrive.

5.1.3 Operations panel

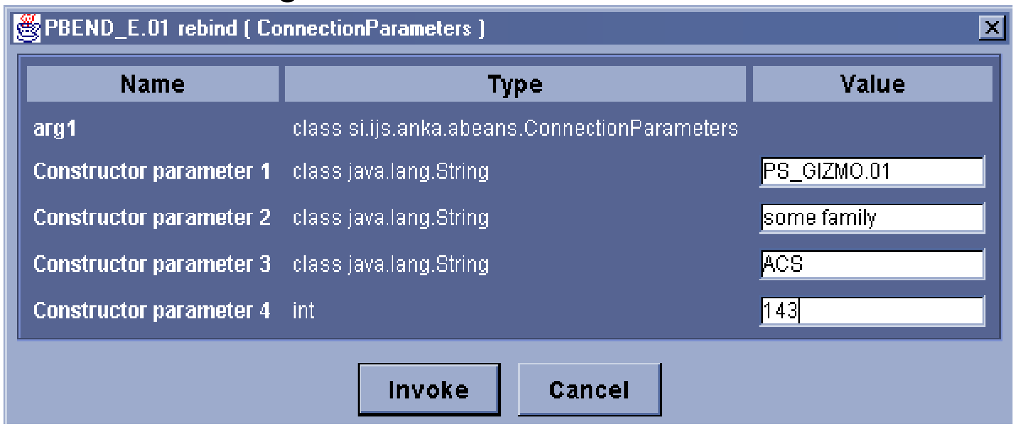

Text output
Trend
Operations
set_value_trigger (double, b
get_value_trigger (<double>
set_timer_trigger (long)
get_timer_trigger (<long>)
suspend ( )
resume ( )
destroy ( )

```
        |booleanValue: false

[3] ----------------------------------------------------------
PBEND_B_01 : readback : create_monitor.suspend
 --> Return value: null
 --> Timeout raised by the engine while waiting for response.

[4] ----------------------------------------------------------
PBEND_B_01 : readback : create_monitor.resume
 --> Return value: null
 --> Timeout raised by the engine while waiting for response.
```

Operations panel shows all the operations of the invocation that produces remote responses. User can invoke operations in a similar manner as in the *operations list* in the main window of Object Explorer. Results are printed in the text area next to the list.

6 OE dialogs

6.1 Parameters dialog


PBEND_E.01 rebind (ConnectionParameters)

Name	Type	Value
arg1	class si.ijs.anka.abeans.ConnectionParameters	
Constructor parameter 1	class java.lang.String	PS_GIZMO.01
Constructor parameter 2	class java.lang.String	some family
Constructor parameter 3	class java.lang.String	ACS
Constructor parameter 4	int	143

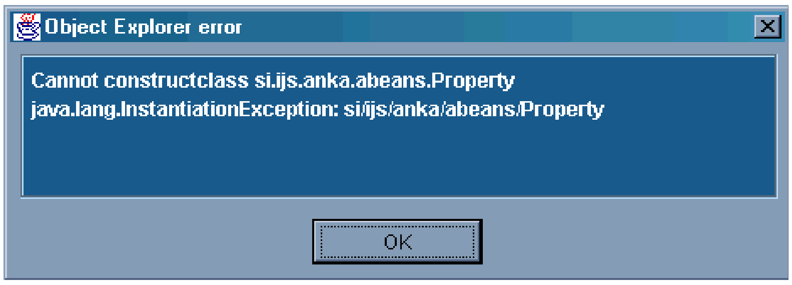
Parameters dialog is shown when an operation, which requires user input, is invoked. It consists of three columns, showing name and type of the parameter, and a component where user specifies the parameter value:

- **primitive types** (except booleans) and strings are entered into a text field as strings and converted into appropriate values.
- **booleans** are entered by checking (true) or unchecking (false) a checkbox
- **enums** (if OE can recognize them) are entered by selecting the appropriate value in a combo box
- **arrays** of primitive types are entered into a text area, each line representing one field in array.
- **structs** and other (simple) **objects** can be provided by entering object's constructor parameters into displayed components.
- interfaces and more complex objects can not be passed as a parameter to operation

To invoke the operation when all parameters are entered, click the *Invoke* button. An error will occur if parameter values are incorrect.

To cancel invocation click the *Cancel* button or close the Parameters dialog

6.2 Error dialog

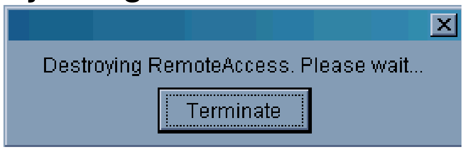


Cannot constructclass si.ijs.anka.abeans.Property
java.lang.InstantiationException: si/ijs/anka/abeans/Property

Error dialog appears when an exception is caught either by the GUI or engine. It displays the error message and exception type. User can select the message in the text area and copy it to clipboard. Exception stack trace is printed to console. To close the error dialog, click the OK button.

Error dialog is modal, so it stops the current thread, preventing multiple dialogs to appear at a time. If another thread raises an error it is also printed out in the same Error dialog.

6.3 RemoteAccessDestroy dialog



Destroying RemoteAccess. Please wait...

This dialog is shown when OE is closing or user has started a new session, causing the engine's RemoteAccess class to disconnect from the remote system. RemoteAccessDestroy dialog prevents from improper destroy and disconnect of the RemoteAccess class, while allowing user to terminate the application normally if the disconnect fails. User can terminate the application or continue with a new session by clicking the Terminate button.

Errors, that occur during the destroy process, do not show an error dialog. The RemoteAccessDestroy dialog indicates how many errors occurred and their exception stack trace is written into the console.

RemoteAccessDestroy dialog closes when the destroy procedure ends, unless there are any errors. In this case user should click the Close button to close the window.

7 Object Explorer Engine

7.1 Overview

The Object Explorer GUI is independent of the control system communication protocol with which the OE communicates. The part of the Object Explorer that depends on the client-server communication architecture (e.g. BACI CORBA, simulator) is called an Engine. All engines implement a common set of interfaces, declared in the si.ijs.acs.objectexplorer.engine package. The only class accessed directly by the OE GUI is the RemoteAccess class, which serves as a factory for other instances of types declared in si.ijs.acs.objectexplorer.engine package. The connection to the remote server is kept alive as long as the OE GUI keeps the RemoteAccess instance alive. In practice, this is as long as the user does not press the “Start Search” button again.

7.2 BACI Engine

BACI engine consists of an implementation of interfaces specified in si.ijs.acs.objectexplorer.engine. BACIRemoteAccess is responsible for establishing a connection to MACI Managers using CORBA. The same class performs the following functionality:

- It uses either the manager and repository specified in the BACI Engine ➠ Manager & IR corbaloc menu or the corbalocs specified as the command line parameters using the –DACS.manager and –DACS.repository options. The corbaloc parameters are always given in the form:

  corbaloc::<server>:port/<ID>

  where ID is Manager for the manager corbaloc and DefaultRepository for the Interface Repository. As soon as these parameters are specified, the BACIRemoteAccess connects to the server and the interface repository.

- After the connection has been established, the BACIRemoteAccess will query the Manager to obtain a list of component names, which the OE GUI will subsequently display in the tree on the left. When an entity is selected in the tree, the BACIRemoteAccess will establish a connection to the remote entity (either a BACI Component or a BACI Property). Furthermore, CORBA introspection will be used to determine BACI compliance and identify BACI design patterns, such as actions, static items etc.

- BACIRemoteAccess will also use CORBA DII to invoke operations on the remote objects and will ask OE GUI to query the users to enter the operation parameters.

- BACIRemoteAccess will finally perform the disconnection and deallocation of the resources, if the OE GUI destroys the engine.

BACI Engine checks also for -Dobjexp.pool_timeout property that defines CORBA DII response poll timeout in milliseconds. (Default is 5000ms.)

In order for the BACI engine to function properly, the following Java components must be properly installed and configured:

1. Manager and repository corbalocs must be specified.

2. OE must be compiled to the jar file and present in the classpath.

3. MACI IDL must be compiled to a jar using IDL to Java compiler and present in the classpath.

4. IDLs for all BACI Components must be compiled to a jar using IDL to Java compiler and present in the classpath

5. Orbacus Interface Repository (irep executable) must be loaded. IDLs of MACI and all BACI Components must be loaded into the repository using the irfeed executable.

Technical note: This is an explanation for the required presence of IDL to Java compiled IDLs for all BACI Components. BACI engine will use these java classes only insofar a BACI Component or Property declares an operation or attribute that passes a complex data type as a parameter. Complex data types supported by BACI engine are structures and enums. In other words, if a Component or a Property passes a user-defined structure as parameter in any operation, the BACI engine will need IDL to Java compiled java stubs for that structure or enum. BACI engine will use Java introspection to load the stubs, query the user to provide instance data for the struct or the enum, instantiate the stub and pass it as an argument to the remote method through CORBA DII. It would be possible to avoid having to provide Java stubs for BACI Components, if CORBA DynAny would be used instead of Java Introspection. Although this is possible, it is quite complicated and considerably slower.

7.3 Value conversion

Object explorer engine supports automatic value conversion. It is made to be completely generic, so even converting e.g. degress (double type) to dd:mm:ss (converter defined structure containing dd, mm, ss) is supported.

An additional property needs to be specified -Dobjexp.converters=<config file>, where value is file name which contains a list of classes implementing Converter instance.

File looks like this (it is located in objexp/test as an example).

```
# list of classes implementing si.ijs.acs.objectexplorer.engine.Converter interface
si.ijs.acs.objectexplorer.engine.BACI.converters.BACIDouble1000Multiplier
si.ijs.acs.objectexplorer.engine.BACI.converters.BACIDegreesToDDMMSSConv
erter
```

In this way objexp is fully extensible.

Conversion can be enabled per inspectable instance - in ACS case: component or property. To enable it right-click over property and select converter in "convert" menu.

Since converters are completely BACI independed, there is a BACIConverterSupport abstract class which helps programmers to write only a minimum set of methods and conversion will be applied on all methods related to property value (all get methods, all set methods, history, set/get_value_trigger, units).

7.4 Supported IDL types, IDL2BACI mapping

As of now, the BACI engine for Object explorer supports the following IDL defined types:

IDL type (IDL type code kind) TCKind	Action taken by OE when packing (as a method argument) / unpacking (as a method return value) arguments of this type
any	**return:** recursively unpacks the contents of any **argument:** not handled by the OE
void	**return & argument:** none
null	**return & argument:** maps to Java null
objref	**return:** unpacks to Java java.lang.Object instance, use introspection to display the accessible fields **argument:** not handled by the OE, unless one of the predefined BACI types (e.g. Callback)
struct, alias, enum	**return:** loads Java stub, extracts packed value by calling "extract" method on the stub. struct retuned by this method is further only introspected and not managed anymore (no conversion is done by objexp) – enums are not converted to its name, octet not converted to short type to handle it as unsigned type. enums are converted to its name using introspection on Java stubs. **argument:** loads Java stub, uses introspection to create a new instance and pass it to the remote call
double	**argument & return:** maps to Java double
float	**argument & return:** maps to Java float

octet	**argument:** maps to Java byte **return:** maps to Java short to handle it as unsigned type
longlong, ulonglong	**argument & return:** maps to Java long
ulong, long	**argument & return:** maps to Java int
short, ushort	**argument & return:** maps to Java short
string	**argument & return:** maps to Java String
char	**argument & return:** maps to Java char
boolean	**argument & return:** maps to Java boolean
other	IllegalArgumentException raised by BACI engine

Note: IDL treats sequences as type alias, because they are defined using the typedef syntax.

BACIIntrospector class uses CORBA introspection to determine design patterns in IDL interfaces that comply with the design patterns defined in the BACI document. BACIIntrospector uses the following rules to determine the nature of the design pattern:

1. If the interface inherits from ACS::Component and is returned by the Manager on the component query, it is considered to represent a BACI Component.

2. If the interface is contained within a BACI Component and inherits from ACS::Property, it is considered to represent the BACI Property.

3. If an interface declares a method, that returns an instance of ACS::Subscription and takes exactly one argument of type inheriting from ACS::Callback, followed by an argument of type ACS::CBDescIn, the method is considered to represent a Invocation, i.e. an action whose duration is unlimited and whose progress will be monitored by a callback (examples are BACI Events and BACI Monitors).

4. If an interface declares an IDL attribute of type readonly, or a method with signature <return value> get<property name>(), and the return value does not

inherit from ACS::Property, this method is considered to be a BACI static item accessor.

5. If an interface declares a method with void return type and variable number of parameters, exactly one of which inherits from ACS::Callback, this one being followed by an ACS::CBDescIn parameter, such method is recognized as a BACI Action.
6. All other methods are invoked and have their parameters packed / unpacked verbatim, with BACI engine making no assumptions about the method behavior.

The contents of the tree are described in Section 3 of this document. BACIIntrospector and BACIRemoteAccess will place, upon making the above analysis:

- BACI Components into the OE tree as Introspectable nodes.
- BACI Properties into the OE tree as Introspectable nodes, children of the corresponding BACI Component nodes. The lifecycle of BACI Property nodes is dependent on the lifecycle of the BACI Component nodes.
- Invocations, as soon as they are created, into the OE tree as SimpleIntrospectable nodes, children of their corresponding Introspectable nodes.
- BACI Actions, methods used verbatim under the Operations list.
- BACI static data item accessors under the Attributes list.

7.5 Callbacks

BACI engine implements BACI callbacks as a CORBA DSI (Dynamic Skeleton Interface) server. This means that OE is capable of unpacking any callback if its interface is present in the interface repository. The callback message will be the name of the method in the callback interface invoked by the server (e.g. “working” or “done”). The callback parameters will be a list of all parameters passed to the callback method. Because DSI is used to unpack callbacks, the OE cannot handle many callbacks set to high refresh rates simultaneously, especially since the rendering of the results also takes a considerable amount of CPU time.