

webMethods EntireX

EntireX Java Wrapper for Natural

Version 9.6

April 2014

This document applies to webMethods EntireX Version 9.6.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1997-2014 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors..

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://documentation.softwareag.com/legal/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

Document ID: EXX-EEXX.JAVAWRAPPER-96-20140628NAT

Table of Contents

I	1	
1	Using Java Wrapper for Natural	3
Prerequisites	4	
Step 1: Start the Java Wrapper for Natural	4	
Step 2: Select the Natural Library (Optional)	5	
Step 3: Select the Natural Subprograms	6	
Step 4: Redesign the Interface for Natural Subprograms (Optional)	8	
Step 5: Configure the Class to be Generated	11	
Generation Result	12	
Preferences	12	
2	Using the Java Wrapper for Natural in Command-line Mode	13
II	Natural to Java Mapping	15
3	Software AG IDL to Java Mapping	17
Mapping IDL Data Types to Java Data Types	18	
Mapping Library Name and Alias	19	
Mapping Program Name and Alias	20	
Mapping Parameter Names	20	
Mapping Fixed and Unbounded Arrays	20	
Mapping Groups and Periodic Groups	21	
Mapping Structures	26	
Mapping the Direction Attributes IN, OUT, and INOUT	31	
Mapping the aligned Attribute	31	
Calling Servers as Procedures or Functions	32	
III		33
4	Writing Applications with the Java Wrapper for Natural	35
5	Delivered Client and Server Examples for Natural	37

I

■ 1 Using Java Wrapper for Natural	3
■ 2 Using the Java Wrapper for Natural in Command-line Mode	13

1 Using Java Wrapper for Natural

■ Prerequisites	4
■ Step 1: Start the Java Wrapper for Natural	4
■ Step 2: Select the Natural Library (Optional)	5
■ Step 3: Select the Natural Subprograms	6
■ Step 4: Redesign the Interface for Natural Subprograms (Optional)	8
■ Step 5: Configure the Class to be Generated	11
■ Generation Result	12
■ Preferences	12

The EntireX Java Wrapper for Natural allows you to generate EntireX Java client interface objects from Natural subprograms in a NaturalONE project in Eclipse. The generated Java client interface objects can be used by Java application developers to access Natural server components, using EntireX/Natural RPC.

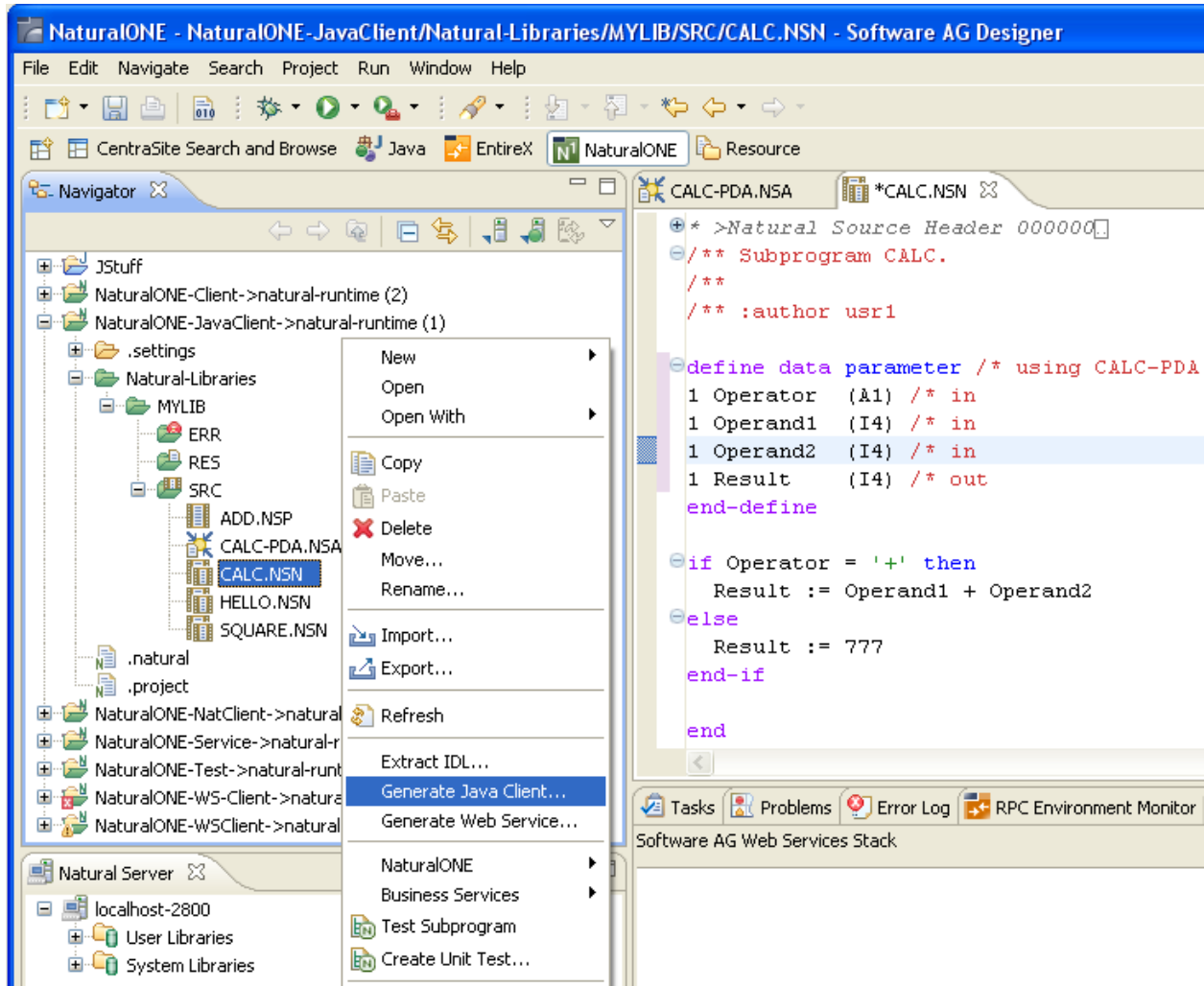
This chapter describes how to generate a client interface object, using Java Wrapper for Natural.

Prerequisites

To use Java Wrapper for Natural you need Software AG Designer with the NaturalONE and EntireX plug-ins installed.

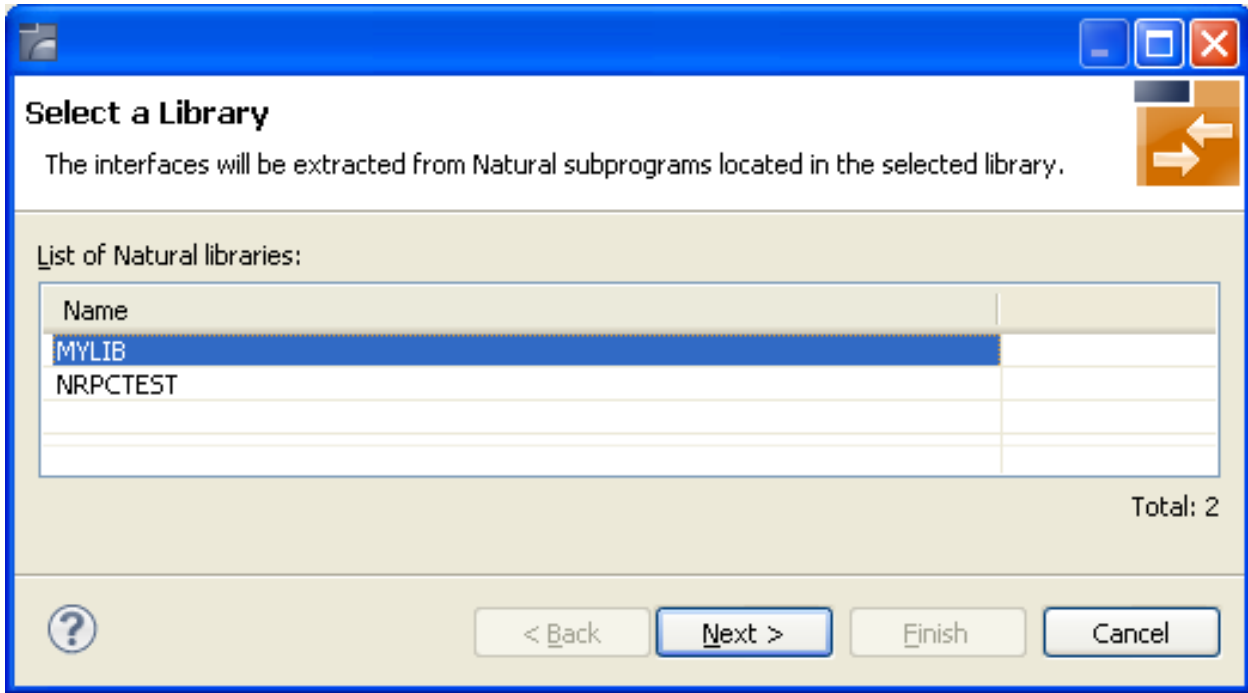
Step 1: Start the Java Wrapper for Natural

To start the Java Wrapper for Natural, select a Natural subprogram (file extension .NSN) located in a library of a NaturalONE project, and from the context menu choose **Generate Java Client...** Alternatively you can start the Java Wrapper for Natural from the context menu of the Natural source folder or any parent folder in the project, including the Natural library and the Project folder.



Step 2: Select the Natural Library (Optional)

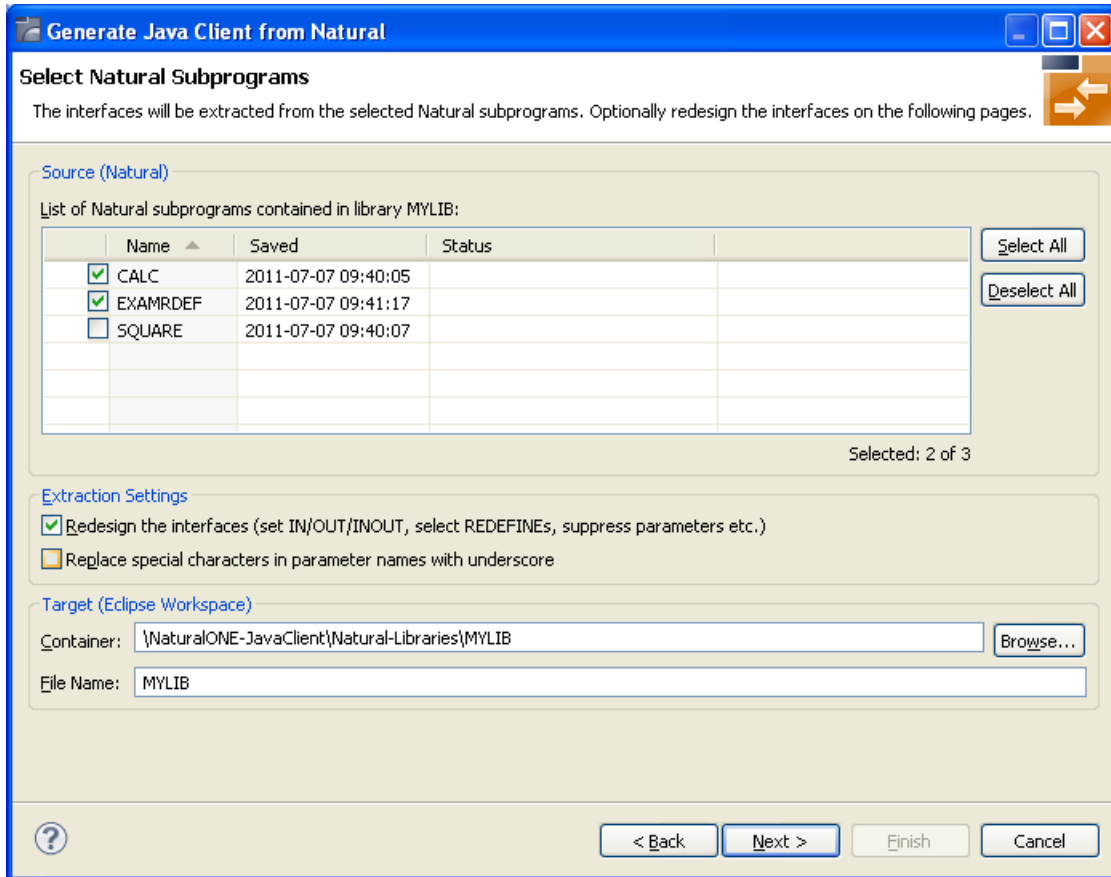
If you have started the wizard from a folder containing multiple Natural libraries, the wizard displays a page showing all available libraries from which you can select one.



Select the Natural library from the list and continue with *Step 3: Select the Natural Subprograms*.

Step 3: Select the Natural Subprograms

The following wizard page provides a list of available Natural subprograms.



In the **Source** pane, select at least one program from the list of Natural subprograms (CALLNATs). You can also choose **Select All** or **Deselect All**.

In the **Extraction Settings** pane, check **Redesign the interfaces** if you want to design the extracted interfaces to the Natural subprograms. The **Next** button will be enabled. See [Step 4: Redesign the Interface for Natural Subprograms \(Optional\)](#). If you do not check **Redesign the interfaces**, see *Natural to IDL Mapping* in the IDL Extractor for Natural documentation for default mappings.

Check **Replace special characters in parameter names by underscore** to substitute the special characters '\$', '#', '&', '@', '/' by underscores. See also *Extracting IDL Parameter Names* in the IDL Extractor for Natural documentation.

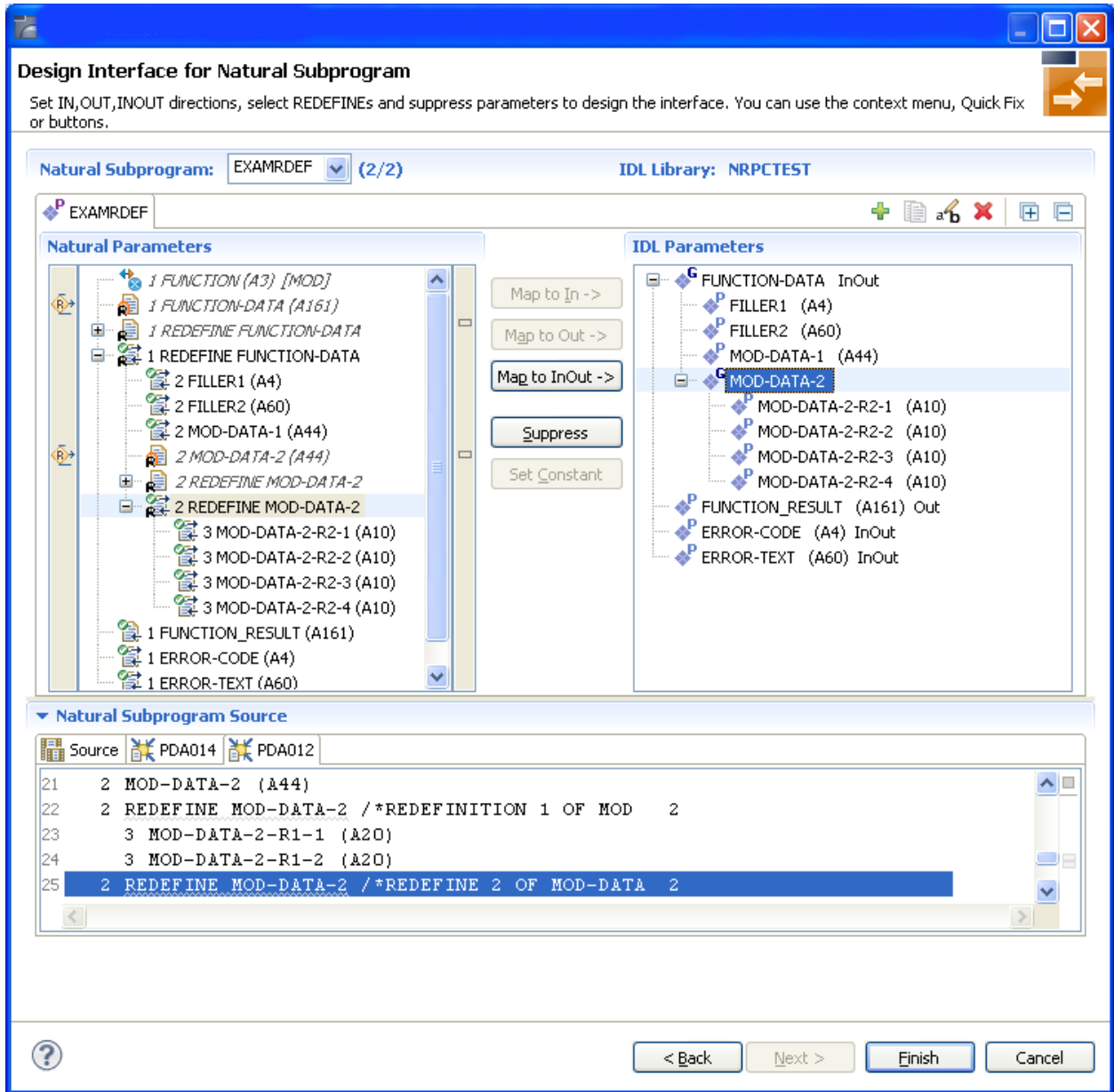
Press **Next** to continue.

- If **Redesign the interfaces** is checked, continue with [Step 4: Redesign the Interface for Natural Subprograms \(Optional\)](#).
- Otherwise continue with [Step 5: Configure the Class to be Generated](#).

Step 4: Redesign the Interface for Natural Subprograms (Optional)

In this step, you can redesign the interface. This includes:

- *Extracting Multiple Interfaces*
- *Extracting Natural REDEFINES*
- *Extracting IDL Directions (IN,OUT,INOUT)*
- *Setting Natural Parameters to Constants*
- *Suppressing Natural Parameters*
- *Renaming a Program*



Use this page for the following tasks:

- Define the direction of parameters in the extracted interface. Choose **Map to In**, **Map to Out** or **Map to InOut** for each parameter on level 1.
- Define which parameters redefined in the Natural PDA are part of the extracted interface. Choose **Map to In**, **Map to Out** or **Map to InOut** for the **REDEFINE** base parameter or any **REDEFINE** path.
- Hide or suppress unneeded parameters in the extracted interface. Choose **Suppress**.
- Set parameters to constants and hide or suppress them in the extracted interface. Choose **Set Constant**.

This page consists of the following main parts:

■ Top line

The top line contains the current Natural subprogram and the IDL library name. The combo box can be used as quick navigation if more than one Natural subprogram is selected.

■ Middle

The middle part contains a tab item for each interface (IDL program) extracted from the Natural subprogram.



Note: It is possible to extract more than one interface (IDL program) from a Natural subprogram. To create, rename and remove interfaces, use the toolbar on the right side of tab folder.

Icon	Function	Description
	Create	Creates a new interface (IDL program) based on the original parameters of the Natural subprogram.
	Duplicate	Creates a new interface (IDL program) based on the current interface (active tab). All modifications of the current interface are copied.
	Rename	Change the name of the current interface (active tab). The name must be unique.
	Remove	Removes the current interface (active tab). At least one interface must exist.
	Expand All	Expands the Natural and IDL tree.
	Collapse All	Collapse the Natural and IDL tree.

■ Middle left

Input pane. The parameters of the Natural subprogram to extract from. For each Natural subprogram parameter you can choose one of the operations **Map to In**, **Map to Out**, **Map to InOut**, **Suppress** and **Set Constant**. Additionally for REDEFINES, a quick fix is available (icons on the left side of the pane) to choose which parameters redefined in the Natural PDA are part of the extracted interface.



Notes:

1. The mapping operations **Map to In**, **Map to Out**, **Map to InOut**, **Suppress** and **Set Constant** are also available in the context menu of the Natural parameter tree.
2. Natural parameters that are suppressed or set to constant in the interface are rendered in italic type. For example, in the screen above, *FUNCTION (A3)* is set to constant; *FILLER1(A4)* and *FILLER2(A60)* are suppressed; *FUNCTION-DATA(A161)* and its first REDEFINE path are implicitly suppressed because the second REDEFINE path with prefix MOD-DATA-2-R2 is selected.
3. The value for Natural parameters set to constant are displayed behind the parameter in the Natural parameter tree (e.g. in the screen above, *FUNCTION (A3) [MOD]*).
4. Natural parameters mapped in the interface are displayed with a green tick (☑).

- **Middle right**

Output pane. The extracted interface (IDL).

- **Bottom**

Reference. The Natural subprogram source and its PDA sources, each displayed in a separate tab.

Tips:

- The panes can be resized.
- To enlarge parameter lists, use the vertical bars on the side.
- You can close the bottom pane if it is not needed by clicking on the triangle next to **Natural Subprogram Source**. In this way, you have more space for viewing the upper panes.

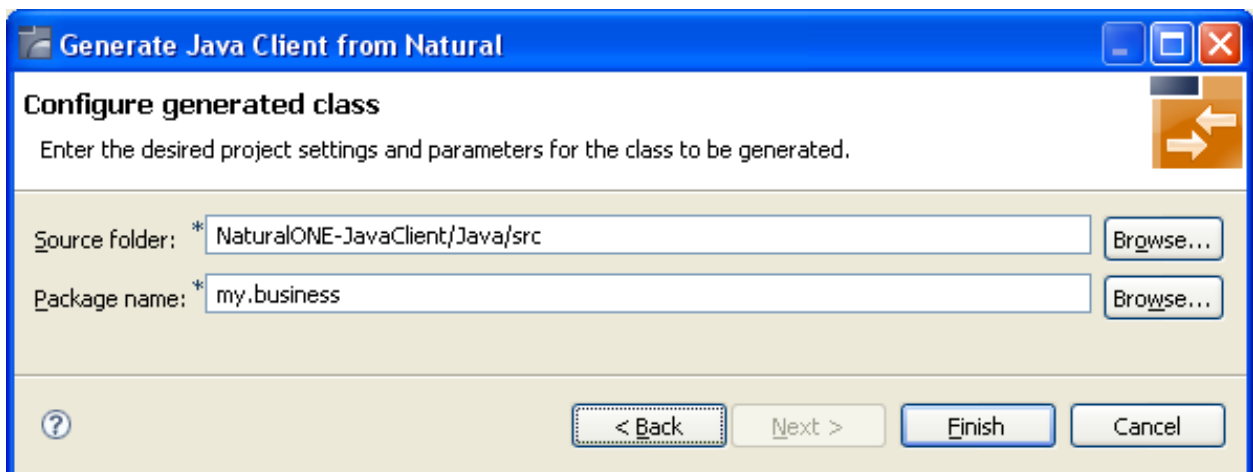
Use the quick navigation or choose **Next** to continue. If multiple Natural subprograms have been selected in the Natural subprogram selection step, redesign the next interface. The amount of subprograms extracted so far is indicated by the fraction next to the title (current/total).

If multiple Natural subprograms have been selected in the Natural subprogram selection step, redesign the next interface. The amount of subprograms extracted so far is indicated by the fraction next to the title (current/total).

If only one Natural subprogram has been selected or no further one has to be redesigned, continue with [Step 5: Configure the Class to be Generated](#).

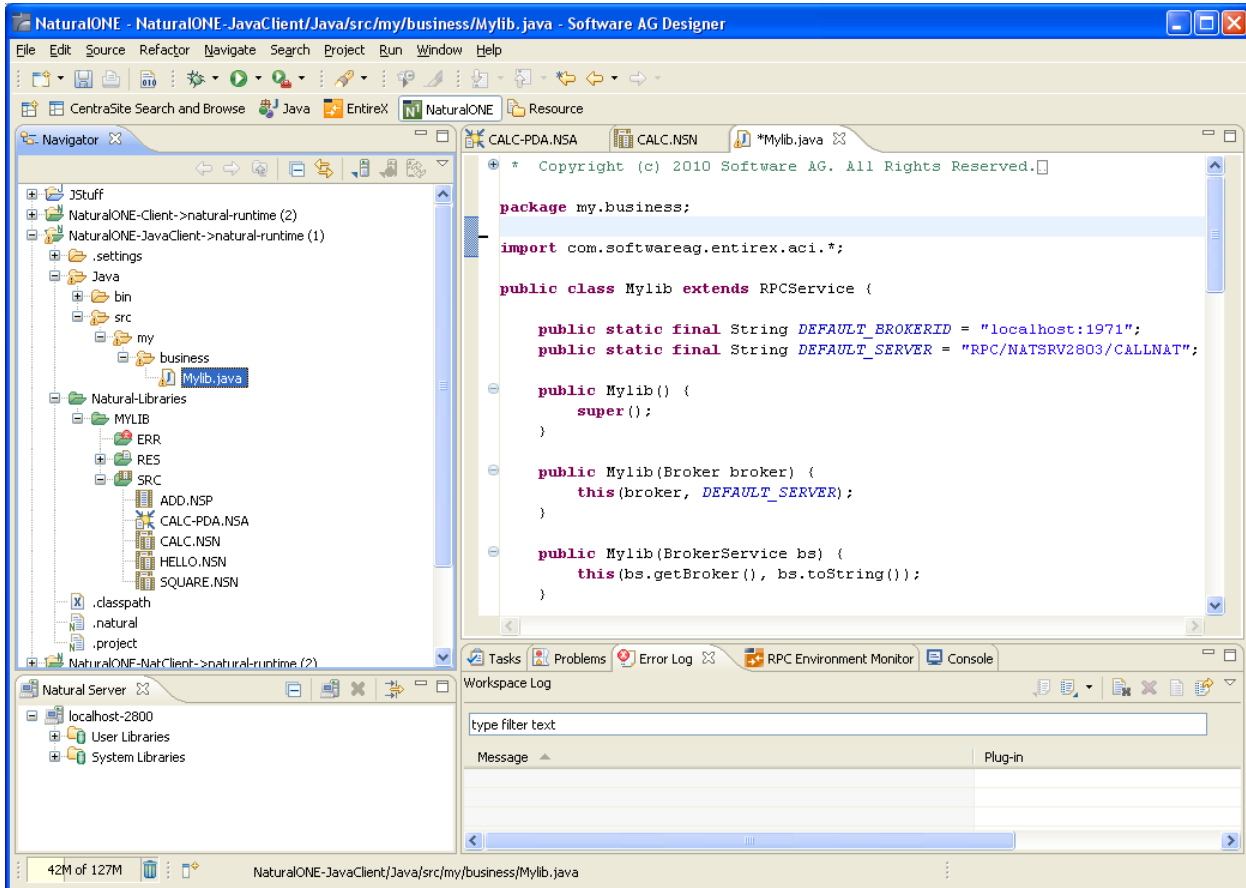
Step 5: Configure the Class to be Generated

The next wizard page asks for the name of the target folder of the generated Java classes, and for a Java package name. By default, the Java sources are generated into a folder *Java/src*. If this does not already exist, it will be created.



Generation Result

The Java interface object is generated and placed into the target folder.



For more information see the NaturalONE documentation.

Preferences

Use the preference page for the IDL Extractor for Natural to manage the default values relevant for [Step 3: Select the Natural Subprograms](#). See *Preferences* under *Using the Software AG IDL Extractor for Natural*.

2 Using the Java Wrapper for Natural in Command-line Mode

Command-line mode is currently not supported.

II

Natural to Java Mapping

This chapter covers the following topics:

Software AG IDL to Java Mapping

See *Natural to IDL Mapping* in the IDL Extractor for Natural documentation.

3 Software AG IDL to Java Mapping

- Mapping IDL Data Types to Java Data Types 18
- Mapping Library Name and Alias 19
- Mapping Program Name and Alias 20
- Mapping Parameter Names 20
- Mapping Fixed and Unbounded Arrays 20
- Mapping Groups and Periodic Groups 21
- Mapping Structures 26
- Mapping the Direction Attributes IN, OUT, and INOUT 31
- Mapping the aligned Attribute 31
- Calling Servers as Procedures or Functions 32

Mapping IDL Data Types to Java Data Types

In the table below, the following metasympols and informal terms are used for the IDL.

- The metasympols [and] surround optional lexical entities.
- The informal term *number* (or *number* [.*number*]) is a sequence of numeric characters, for example 123.

Software AG IDL	Description	Java Data Types	Note
<i>A</i> <i>number</i>	Alphanumeric	String	1, 3
AV	Alphanumeric variable length	String	
AV[<i>number</i>]	Alphanumeric variable length with maximum length	String	1
B <i>number</i>	Binary	byte[]	1, 6
BV	Binary variable length	byte[]	
BV[<i>number</i>]	Binary variable length with maximum length	byte[]	1
D	Date	java.util.Date	5
F4	Floating point (small)	float	2
F8	Floating point (large)	double	2
I1	Integer (small)	byte	
I2	Integer (medium)	short	
I4	Integer (large)	int	
K <i>number</i>	Kanji	String	1
KV	Kanji variable length	String	
KV[<i>number</i>]	Kanji variable length with maximum length	String	1
L	Logical	boolean	
N <i>number</i> [. <i>number</i>]	Unpacked decimal	java.math.BigDecimal	4
NU <i>number</i> [. <i>number</i>]	Unpacked decimal unsigned	java.math.BigDecimal	4
P <i>number</i> [. <i>number</i>]	Packed decimal	java.math.BigDecimal	4
PU <i>number</i> [. <i>number</i>]	Packed decimal unsigned	java.math.BigDecimal	4
T	Time	java.util.Date	5
U <i>number</i>	Unicode	String	7
UV	Unicode variable length	String	7
UV <i>number</i>	Unicode variable length with maximum length	String	7



Notes:

1. The field length is given in bytes.

2. If floating-point data types are used, rounding errors can occur. Therefore, the values of sender and receiver might differ slightly.
3. If you use the value null (null pointer) as an input parameter (for IN and INOUT parameters) for type A, a blank string will be used.
4. If you use the value null (null pointer) as an input parameter (for IN and INOUT parameters) for types N/P, the value 0 (or 0.0) will be used.
5. If you use the value null (null pointer) as an input parameter (for IN and INOUT parameters) for types D/T, the current date/time will be used. You change this with the property `entirex.marshal.date`. Setting `entirex.marshal.date=null` will map the value null to the invalid date 0000-01-01 of the RPC marshalling. This is the invalid date value in Natural, too. With this setting the invalid date as an output parameter will be mapped to null. The default is to map the invalid date to 0001-01-01.
6. If you use the value null (null pointer) as an input parameter (for IN and INOUT parameters) for type B, all binary values will be set to zero.
7. The length is given in 2-byte Unicode code units following the Unicode standard UTF-16. The maximum length is 805306367 code units.

Please note also hints and restrictions on the Software AG IDL data types valid for all programming language bindings. See *IDL Data Types* under *Software AG IDL File* in the IDL Editor documentation.

Mapping Library Name and Alias

The library name as specified in the IDL file is sent from a client to the server. Special characters are not replaced. The library alias is not sent to the server.

In the RPC server, the IDL library name sent may be used to locate the target server. See *Locating and Calling the Target Server* in the platform-specific administration or RPC server documentation.

The library name as given in the library definition of the IDL file is mapped to the class name of the generated Java classes. See `library-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation. For the server interface object, the names of the class are composed as `library name Interface Object` and `library name Server`. For the client interface object, no suffix is appended. When the class names are built, the library name is capitalized to match Java naming conventions.

The special characters '#' and '-' in the library name are replaced by the character '_'.

If there is an alias for the library name in the `library-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation, this alias is used as is to form the client class name. Therefore, this alias must be a valid Java class name. On the server side, the alias is used as is to form the class name of the server class.

Example:

- library name Hu#G-O is converted to Hu_g_o

Mapping Program Name and Alias

The program name is sent from a client to the server. Special characters are not replaced. The program alias is not sent to the server.

In the RPC server, the IDL program name sent is used to locate the target server. See *Locating and Calling the Target Server* in the platform-specific administration or RPC server documentation.

The program name as given in the `program-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation of the IDL file is mapped to method names within the generated Java classes. To match Java naming conventions the program name is converted to lowercase.

The special characters '#' and '-' in the program name are replaced by the character '_'.

If there is an alias for the program name in the `program-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation, this alias is used as is for the method name. Therefore, this alias must be a valid Java method name. On the server side, the alias is used as is for the method name in the server class.

Mapping Parameter Names

The parameter names are mapped to fields inside the classes (see *Mapping the Direction Attributes IN, OUT, and INOUT* in the Java Wrapper documentation).

Example:

- parameter name Hu#G-O is converted to hu_g_o

Mapping Fixed and Unbounded Arrays

Arrays in the IDL file are mapped to Java arrays. If an array value does not have the correct number of dimensions or elements, this will result in a `NullPointerException` or an `ArrayIndexOutOfBoundsException`. If you use the value `null` (null pointer) as an input parameter (for `IN` and `INOUT` parameters), an array will be instantiated.

Mapping Groups and Periodic Groups

Groups (structures) in the IDL file are mapped to inner classes. If the Bean-compliant generation mode is used, they are mapped to normal classes in their own files. The group members (structure fields) are implemented as public fields of the inner class. If the bean-compliant generation is used, the members (structure fields) are implemented as private fields with getter and setter methods.

Example

The following example shows how to program with groups in a Java client and server. The IDL program consists of three groups, each with the same fields, but with different directions. The client shows how to initialize the fields in the groups for the In and InOut parameters and how to get the results from the Out and InOut parameters. The server part shows only the implemented server method, not the other parts of the generated server skeleton. The server just moves the data from the In parameters to the Out parameters and fills the gaps. We assume that `ClientGroup.class` and the client interface object `Libgroup.class` are in the same folder. To compile and run the client and the server you need the *entirex.jar*. For the server we assume that `LibgroupServer.class` and `LibgroupStub.class` are in the same folder and this folder is in the classpath of the EntireX Java RPC Server.

IDL

```
library 'LibGroup' is
  program 'Program1' is
    define data parameter
      1 Group1    (/3)    In Out
        2 Field01  (A10)
        2 Field02  (N2)
        2 Field03  (I4)
      1 Group2    (/1)    In
        2 Field01  (A10)
        2 Field02  (N2)
        2 Field03  (I4)
      1 Group3    (/2)    Out
        2 Field01  (A10)
        2 Field02  (N2)
        2 Field03  (I4)
    end-define
```

Client

```

import com.softwareag.entirex.aci.Broker;
import com.softwareag.entirex.aci.BrokerException;
import java.math.BigDecimal;

public class ClientGroup {
    public static void main(String[] args) {
        try {
            Broker broker = new Broker(Libgroup.DEFAULT_BROKERID, "User1");
            broker.logon();
            // create the wrapper object.
            Libgroup lib = new Libgroup(broker, Libgroup.DEFAULT_SERVER);
            // /*
            // * Using the old style:
            // * Get the reference for group1 from wrapper object and
            // * fill group1 with data. Since group1 is InOut, there exists a
            // * reference.
            // */
            // Group1[] group1 = lib.getGroup1();
            // for (int i = 0; i < group1.length; i++) {
            //     // create a new instance of each array element of group1.
            //     group1[i] = new Group1();
            //     // fill the data in each field.
            //     group1[i].setField01("group1 " + i);
            //     group1[i].setField02(new BigDecimal(i));
            //     group1[i].setField03(2 * i);
            // }
            /*
            * Fill the group1 parameters, using the new methods for indexed access.
            */
            Group1[] group1 = lib.getGroup1();
            for (int i = 0; i < group1.length; i++) {
                Group1 group = new Group1();
                group.setField01("group1 " + i);
                group.setField02(new BigDecimal(i));
                group.setField03(2 * i);
                lib.setGroup1(i, group);
            }

            /*
            * Create an instance for group2. There is no reference for group2
            * since this is an In parameter. Fill group2 with data.
            */
            Group1[] group2 = new Group1[1];
            for (int i = 0; i < group2.length; i++) {
                // create a new instance of each array element of group2.
                group2[i] = new Group1();
                // fill the data in each field.
                group2[i].setField01("group2 " + i);
                group2[i].setField02(new BigDecimal(i));
            }
        }
    }
}

```

```

        group2[i].setField03(2 * i);
    }
    // do the RPC.
    lib.program1(group2);

    // /*
    // * Using the old style:
    // * We can use the reference group1, it is not modified.
    // */
    // for (int i = 0; i < group1.length; i++) {
    //     // get the data from the group and print.
    //     System.out.println("Result of Program1; group1[" + i + "] "
    //         + group1[i].getField01() + ", " + group1[i].getField02() + ↵
", "
        //         + group1[i].getField03());
    // }
    /*
    * Retrieve the group1 elements, using the new indexed access method.
    */
    for (int i = 0; i < 3; i++) {
        // get the data from the group and print.
        System.out.println("Result of Program1; group1[" + i + "] "
            + lib.getGroup1(i).getField01() + ", "
            + lib.getGroup1(i).getField02() + ", "
            + lib.getGroup1(i).getField03());
    }

    // /*
    // * Using the old style:
    // * Get the reference for group3. group3 is Out.
    // */
    // Group1[] group3 = lib.getGroup3();
    // for (int i = 0; i < group3.length; i++) {
    //     // get the data from the group and print.
    //     System.out.println("Result of Program1; group3[" + i + "] "
    //         + group3[i].getField01() + ", " + group3[i].getField02() + ↵
", "
        //         + group3[i].getField03());
    // }
    /*
    * Retrieve the group3 elements, using the new indexed access method.
    */
    for (int i = 0; i < 2; i++) {
        // get the data from the group and print.
        System.out.println("Result of Program1; group3[" + i + "] "
            + lib.getGroup3(i).getField01() + ", "
            + lib.getGroup3(i).getField02() + ", "
            + lib.getGroup3(i).getField03());
    }

    broker.logoff();
} catch (BrokerException excep) {
    excep.printStackTrace ();
}

```

```

}
}

```

Client Group (Bean-compliant)

```

import com.softwareag.entirex.aci.Broker;
import com.softwareag.entirex.aci.BrokerException;
import java.math.BigDecimal;

public class ClientGroup {
    public static void main(String[] args) {
        try {
            Broker broker = new Broker(Libgroup.DEFAULT_BROKERID, "User1");
            broker.logon();
            // create the wrapper object.
            Libgroup lib = new Libgroup(broker, Libgroup.DEFAULT_SERVER);
            /* Get the reference for group1 from wrapper object and
             * fill group1 with data. Since group1 is InOut, there exists a
             * reference.
             */
            Group1[] group1 = lib.getGroup1();
            for (int i = 0; i < group1.length; i++) {
                // create a new instance of each array element of group1.
                group1[i] = new Group1();
                // fill the data in each field.
                group1[i].setField01("group1 " + i);
                group1[i].setField02(new BigDecimal(i));
                group1[i].setField03(2 * i);
            }
            /** Create an instance for group2. There is no reference for group2
             * since this is an In parameter. Fill group2 with data.
             */
            Group1[] group2 = new Group1[1];
            for (int i = 0; i < group2.length; i++) {
                // create a new instance of each array element of group2.
                group2[i] = new Group1();
                // fill the data in each field.
                group2[i].setField01("group2 " + i);
                group2[i].setField02(new BigDecimal(i));
                group2[i].setField03(2 * i);
            }
            // do the RPC.
            lib.program1(group2);
            // We can use the reference group1, it is not modified.
            for (int i = 0; i < group1.length; i++) {
                // get the data from the group and print.
                System.out.println("Result of Program1; group1[" + i + "] "
                    + group1[i].getField01() + ", " + group1[i].getField02() + ", "
                    + group1[i].getField03());
            }
        }
    }
}

```

```

        // Get the reference for group3. group3 is Out.
        Group1[] group3 = lib.getGroup3();
        for (int i = 0; i < group3.length; i++) {
            // get the data from the group and print.
            System.out.println("Result of Program1; group3[" + i + "] "
                + group3[i].getField01() + ", " + group3[i].getField02() + ", "
                + group3[i].getField03());
        }
        broker.logoff();
    } catch (BrokerException excep) {
        excep.printStackTrace ();
    }
}
}
}

```

Server

```

public void program1 (LibgroupServer.Program1Group2[] group2) {
    /*
     * Program1Group1 is InOut
     * Program1Group2 is In
     * Program1Group3 is Out
     * Move the values from Program1Group2 to Program1Group1 and move the
     * value from Program1Group1 to Program1Group3.
     */
    int length = Math.min(program1Group1.length, program1Group3.length);
    for (int i = 0; i < length; i++) {
        if (program1Group3[i] == null)
            program1Group3[i] = new Program1Group3();
        program1Group3[i].field01 = program1Group1[i].field01;
        program1Group3[i].field02 = program1Group1[i].field02;
        program1Group3[i].field03 = program1Group1[i].field03;
    }
    for (int i = length; i < program1Group3.length; i++) {
        if (program1Group3[i] == null)
            program1Group3[i] = new Program1Group3();
        program1Group3[i].field01 = "New Text " + i;
        program1Group3[i].field02 = new BigDecimal(10);
        program1Group3[i].field03 = 100 + i;
    }
    // move the values from Program1Group1 to Program1Group3.
    length = Math.min(group2.length, program1Group1.length);
    for (int i = 0; i < length; i++) {
        if (program1Group1[i] == null)
            program1Group1[i] = new Program1Group1();
        program1Group1[i].field01 = group2[i].field01;
        program1Group1[i].field02 = group2[i].field02;
        program1Group1[i].field03 = group2[i].field03;
    }
    for (int i = length; i < program1Group1.length; i++) {

```

```

        if (program1Group1[i] == null)
            program1Group1[i] = new Program1Group1();
        program1Group1[i].field01 = "New Text " + i;
        program1Group1[i].field02 = new BigDecimal(10);
        program1Group1[i].field03 = 100 + i;
    }
}

```

Mapping Structures

Structures are mapped like Groups. See *Mapping Groups and Periodic Groups* in the Java Wrapper documentation.

Example

The following example shows how to program with structures in a Java client and server. The structures are mapped to inner classes of the interface objects; if Bean-compliant generation is used, the structures are mapped to normal classes in their own file. The IDL program consists of one structure that is used with different directions. In the example above for the groups we have the same fields in each group. This example shows how to simplify this by using a structure. The structure is defined outside the program and references to the structure can be used several times in different programs. The client shows how to initialize the fields in the references of the structure for the In and InOut parameters and how to get the results from the Out and InOut parameters. The server part shows only the implemented server method, not the other parts of the generated server skeleton. The server just moves the data from the In parameters to the Out parameters and fills the gaps. We assume that `ClientStrct.class` and the client interface object `Libstrct.class` are in the same folder. To compile and run the client and the server you need the `entirex.jar`. For the server we assume that `LibstrctServer.class` and `LibstrctStub.class` are in the same folder and this folder is in the classpath of the EntireX Java RPC Server.

IDL

```

library 'LibStrct' is
    struct 'Struct1' is
        define data parameter
            1 Field01 (A10)
            1 Field02 (N2)
            1 Field03 (I4)
        end-define

    program 'Program1' is
        define data parameter
            1 Ref1 ('Struct1'/3) In Out
            1 Ref2 ('Struct1'/1) In

```

```
1 Ref3 ('Struct1'/2) Out
end-define
```

Client

```
import com.softwareag.entirex.aci.Broker;
import com.softwareag.entirex.aci.BrokerException;
import java.math.BigDecimal;

public class ClientStrct {
    public static void main(String[] args) {
        try {
            Broker broker = new Broker(Libstrct.DEFAULT_BROKERID, "User1");
            broker.logon();
            // create the wrapper object.
            Libstrct lib = new Libstrct(broker, Libstrct.DEFAULT_SERVER);
            /* create a struct object (as defined in the wrapper object) for the
             * InOut parameter struct1.
             */
            Struct1[] struct1 = new Struct1[3];
            // /*
            // * Using the old style:
            // * fill the struct object with data.
            // */
            for (int i = 0; i < struct1.length; i++) {
                // // create a new array element.
                // struct1[i] = new Struct1();
                // struct1[i].setField01("struct1 ");
                // struct1[i].setField02(new BigDecimal(4 + i));
                // struct1[i].setField03(i);
                // }
                // // set the struct object in the wrapper object
                // lib.setRef1 (struct1);
            /*
             * Fill the struct1 parameters, using the new methods for indexed access.
             */
            for (int i = 0; i < struct1.length; i++) {
                Struct1 struct = new Struct1();
                struct.setField01("struct1 ");
                struct.setField02(new BigDecimal(4 + i));
                struct.setField03(i);
                lib.setRef1(i, struct);
            }
            /* create a struct object (as defined in the wrapper object) for the
             * In parameter struct2.
             */
            Struct1[] struct2 = new Struct1[1];
            for (int i = 0; i < struct2.length; i++) {
                // create a new array element.
                struct2[i] = new Struct1();
                struct2[i].setField01("struct2 ");
            }
        }
    }
}
```

```

        struct2[i].setField02(new BigDecimal(4 + i));
        struct2[i].setField03(i);
    }

    // do the RPC.
    lib.program1(struct2);

    // /*
    // * Using the old style:
    // * get the data from the InOut parameter struct1.
    // */
    // for (int i = 0; i < struct1.length; i++) {
    //     // get the data from the struct and print.
    //     System.out.println("Result of Program1, struct1[" + i + "] "
    //         + struct1[i].getField01() + ", " + struct1[i].getField02() ←
+ ", "
    //         + struct1[i].getField03());
    // }
    /*
    * Retrieve the ref1 elements, using the new indexed access method.
    */
    for (int i = 0; i < 3; i++) {
        // get the data from the struct and print.
        System.out.println("Result of Program1, struct1[" + i + "] "
            + lib.getRef1(i).getField01() + ", "
            + lib.getRef1(i).getField02() + ", "
            + lib.getRef1(i).getField03());
    }
    // /*
    // * Using the old style:
    // * get the struct object for the Out parameter struct3.
    // */
    // Struct1[] struct3 = lib.getRef3();
    // // get the data from the Out parameter struct3.
    // for (int i = 0; i < struct3.length; i++) {
    //     // get the data from the struct and print.
    //     System.out.println("Result of Program1, struct3[" + i + "] "
    //         + struct3[i].getField01() + ", " + struct3[i].getField02() + ", "
    //         + struct3[i].getField03());
    // }
    /*
    * Retrieve the ref3 elements, using the new indexed access method.
    */
    for (int i = 0; i < 2; i++) {
        // get the data from the struct and print.
        System.out.println("Result of Program1, struct3[" + i + "] "
            + lib.getRef3(i).getField01() + ", "
            + lib.getRef3(i).getField02() + ", "
            + lib.getRef3(i).getField03());
    }

    broker.logoff();
} catch (BrokerException excep) {

```



```

        excep.printStackTrace ();
    }
}

```

ClientStrct (Bean-compliant)

```

import com.softwareag.entirex.aci.Broker;
import com.softwareag.entirex.aci.BrokerException;
import java.math.BigDecimal;

public class ClientStrct {
    public static void main(String[] args) {
        try {
            Broker broker = new Broker(Libstrct.DEFAULT_BROKERID, "User1");
            broker.logon();
            // create the wrapper object.
            Libstrct lib = new Libstrct(broker, Libstrct.DEFAULT_SERVER);
            /* create a struct object (as defined in the wrapper object) for the
             * InOut parameter struct1.
             */
            Struct1[] struct1 = new Struct1[3];
            // fill the struct object with data.
            for (int i = 0; i < struct1.length; i++) {
                // create a new array element.
                struct1[i] = new Struct1();
                struct1[i].setField01("struct1 ");
                struct1[i].setField02(new BigDecimal(4 + i));
                struct1[i].setField03(i);
            }
            /* create a struct object (as defined in the wrapper object) for the
             * In parameter struct2.
             */
            Struct1[] struct2 = new Struct1[1];
            for (int i = 0; i < struct2.length; i++) {
                // create a new array element.
                struct2[i] = new Struct1();
                struct2[i].setField01("struct2 ");
                struct2[i].setField02(new BigDecimal(4 + i));
                struct2[i].setField03(i);
            }
            // set the struct object in the wrapper object
            lib.setRef1 (struct1);
            // do the RPC.
            lib.program1(struct2);
            // get the struct object for the Out parameter struct3.
            Struct1[] struct3 = lib.getRef3();
            // get the data from the InOut parameter struct1.
            for (int i = 0; i < struct1.length; i++) {
                // get the data from the struct and print.
                System.out.println("Result of Program1, struct1[" + i + "] "

```

```

        + struct1[i].getField01() + ", " + struct1[i].getField02() + "
", "
        + struct1[i].getField03());
    }
    // get the data from the Out parameter struct3.
    for (int i = 0; i < struct3.length; i++) {
        // get the data from the struct and print.
        System.out.println("Result of Program1, struct3[" + i + "] "
            + struct3[i].getField01() + ", " + struct3[i].getField02() + "
", "
            + struct3[i].getField03());
    }
    broker.logoff();
} catch (BrokerException excep) {
    excep.printStackTrace ();
}
}
}

```

Server

```

public void program1 (Struct1[] ref2) {
    /*
     * Program1Group1 is InOut
     * Program1Group2 is In
     * Program1Group3 is Out
     * Move the values from Program1Group2 to Program1Group1 and move the
     * value from Program1Group1 to Program1Group3.
     */
    int length = Math.min(program1Ref1.length, program1Ref3.length);
    for (int i = 0; i < length; i++) {
        if (program1Ref3[i] == null)
            program1Ref3[i] = new Struct1();
        program1Ref3[i].field01 = program1Ref1[i].field01;
        program1Ref3[i].field02 = program1Ref1[i].field02;
        program1Ref3[i].field03 = program1Ref1[i].field03;
    }
    for (int i = length; i < program1Ref3.length; i++) {
        if (program1Ref3[i] == null)
            program1Ref3[i] = new Struct1();
        program1Ref3[i].field01 = "New Text " + i;
        program1Ref3[i].field02 = new BigDecimal(10);
        program1Ref3[i].field03 = 100 + i;
    }

    length = Math.min(ref2.length, program1Ref1.length);
    for (int i = 0; i < length; i++) {
        if (program1Ref1[i] == null)
            program1Ref1[i] = new Struct1();
        program1Ref1[i].field01 = ref2[i].field01;
        program1Ref1[i].field02 = ref2[i].field02;
    }
}

```

```

        program1Ref1[i].field03 = ref2[i].field03;
    }
    for (int i = length; i < program1Ref1.length; i++) {
        if (program1Ref1[i] == null)
            program1Ref1[i] = new Struct1();
        program1Ref1[i].field01 = "New Text " + i;
        program1Ref1[i].field02 = new BigDecimal(10);
        program1Ref1[i].field03 = 100 + i;
    }
}

```

Mapping the Direction Attributes IN, OUT, and INOUT

The IDL syntax allows you to define parameters as `IN` parameters, `OUT` parameters, or `IN OUT` parameters (which is the default if nothing is specified). This direction specification is reflected in the generated Java interface object as follows:

- `IN` parameters are sent from the RPC client to the RPC server. `IN` parameters are implemented as parameters of the generated method.
- `OUT` parameters are sent from the RPC server to the RPC client. `OUT` parameters are implemented as read-only properties. A `getMethod` is generated for each `OUT` parameter.
- `INOUT` parameters are sent from the RPC client to the RPC server and then back to the RPC client. `INOUT` parameters are implemented as properties. A `setMethod` and a corresponding `getMethod` is generated for each `INOUT` parameter.

Note that only the direction information of the top-level fields (level 1) is relevant. Group fields always inherit the specification from their parent. A different specification is ignored.

See the `attribute-list` under *Software AG IDL Grammar* in the *IDL Editor* documentation for the syntax on how to describe attributes in the IDL file and refer to the `direction` attribute.

Mapping the aligned Attribute

The `aligned` attribute is not relevant for the programming language Java. However, a Java client can send the `aligned` attribute to an EntireX RPC server, where it might be needed.

See the `attribute-list` under *Software AG IDL Grammar* in the *IDL Editor* documentation for the syntax on how to describe attributes in the IDL file and refer to the `aligned` attribute.

Calling Servers as Procedures or Functions

The IDL syntax allows definition of procedures only. It does not have the concept of a function. A function is a procedure which, in addition to the parameters, returns a value. Procedures and functions are transparent between clients and server, i.e. a client using a function can call a server implemented as a procedure and vice versa. In Java a procedure corresponds to a method with result type `void`, a function returns a value of some type.

It is possible to treat the `OUT` parameter of a procedure as the return value of a function. The Java Wrapper generates a method with a non-void result type when the following conditions are met:

- the last parameter of the procedure definition is of type `OUT`;
- this last parameter of the procedure definition has the name `Function_Result`. The name `Function_Result` is not case-sensitive.

Of course, in this case `getMethod` is not generated for this `OUT` parameter.

As an example, see the Java Wrapper example that comes with EntireX.

III

■ 4 Writing Applications with the Java Wrapper for Natural	35
■ 5 Delivered Client and Server Examples for Natural	37

4 Writing Applications with the Java Wrapper for Natural

Following is a simple Java client application that uses the generated Java interface object's `calc` method to implement an add operation:

```
package my.business;

import com.softwareag.entirex.aci.Broker;

public class Add {

    private Mylib m = new Mylib(new Broker(Mylib.DEFAULT_BROKERID, "user"), ↵
    Mylib.DEFAULT_SERVER);

    public int add(int operand1, int operand2) throws Exception {
        m.setNaturalLogon(true); // NaturalONE RPC server requires logon
        m.calc("+", operand1, operand2);
        return m.getCalcResult();
    }

    public static void main(String[] args) {
        try {
            System.out.println("Result is: " + new Add().add(1,2));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

5

Delivered Client and Server Examples for Natural

See the readme files of the following examples for more information:

- `<EntireX>/examples/RPC/basic/example/NaturalServer`
- `<EntireX>/examples/RPC/basic/example/JavaClient`
- `<EntireX>/examples/RPC/reliable/NaturalServer`
- `<EntireX>/examples/RPC/reliable/JavaClient`

