Infor LN Studio Integration Development Guide

Release 10.7.x

Publication Information

Release: Infor LN 10.7.x
Publication Date: July 5, 2021
Document code: ln_10.7.x_lnstudiointdg__en-us

Contents

About this guide

This Developer's guide describes how you can develop and deploy business interfaces for LN application servers, using the LN Studio.

An elementary understanding is needed to use this guide; you require general knowledge about Business Data Entity (BDE) interfaces, Oagis Business Object Documents (BODs) and the way Infor LN or Baan software is structured.

This document is valid for the following product versions:

- LN Studio 8.4.2 and 8.5 or later.
- Enterprise Server 8.4.2 or later.

Note: A runtime from LN Studio requires at least Enterprise Server 8.4.2. Therefore, when using the LN Studio, do not generate into LN FP4 or lower if the runtime is also delivered to customers or sites that are still on Enterprise Server 8.4.1 or lower.

This document contains the following development information:

- Chapter 1: "Introduction". A short explanation about Abbreviations and Terminology used in this document.
- Chapter 2: "Creating a Business Interface Implementation for LN" describes how to use the LN Studio to create business interface implementations specifically for LN application servers and how to generate the corresponding business objects in LN
- Chapter 3: "Using Hooks for LN", explains how to write specific logic in hooks.
- Chapter 4: "Method-Specific Guidelines", discusses guidelines to take into account when implementing standard methods.
- Chapter 5: "Publishing and Receiving Events", describes how to implement event publishing methods and how to adapt an LN application package to allow it to publish application-specific events for a business object.
- Chapter 6: "Modeling BIIs for Complex Cases", is about how to implement business objects in specific complex situations.
- Chapter 7: "Import from BOR": how to migrate existing LN business objects from the BOR to the LN Studio.
- Chapter 8: Implementing OAGIS BODs for LN", explains how to model and implement business interfaces for OAGIS BODs.

Caution: With Business Studio 8.4.2 and Enterprise Server 8.4.2, BOD-related functionality is not generally available. Do not create BODs or BOD-based integrations without prior written consent from Infor. No support will be given on the use of BODs (unless agreed otherwise).

Contacting Infor

If you have questions about Infor products, go to Infor Concierge at https://concierge.infor.com/ and create a support incident.

The latest documentation is available from docs.infor.com or from the Infor Support Portal. To access documentation on the Infor Support Portal, select **Search > Browse Documentation**. We recommend that you check this portal periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1: Introduction

Abbreviations and Terminology

The following abbreviations are used in this document:

Term	Definition
BDE	Business Data Entity
BI	Business Interface
BID	Business Interface Definition
BII	Business Interface Implementation
BOD	Business Object Document
BOL	Business Object Layer
BOR	Business Object Repository
LN	The LN enterprise resource planning product
FP	Feature Pack
VRC	Version - Release - Customer, which together define the edition of a software component in LN
XSD	XML Schema Definition

Some crucial terms used in this document may need some explanation in advance.

A BID (Business Interface Definition) is the representation of an interface as modeled in the LN Studio. A BID for example specifies the structure of a sales order object having a header and lines, each having multiple attributes. Additionally it defines the methods that can be invoked for the sales order or the events that can be sent to or from a sales order instance.

The BII (Business Interface Implementation) describes the implementation of the BID for an application such as LN It contains the mapping of the interface to database tables, forms and business logic as existing in LN.

A business object is the actual implementation of a BII in the LN application. It is generated from the LN Studio, based on a BII and the corresponding BID.

Two interface variants exist for BIDs and consequently for business objects:

- BDE interface.

The business object offers an interface that adheres to the Business Data Entity Implementation Standard and Event Management Standard.

- BOD interface.

  The business object can send and receive OAGIS BODs (Business Object Documents). Note that an LN business object that has a BOD interface can also offer protected BDE methods.

Before Enterprise Server 8.3, business objects were modeled and generated from the BOR (Business Object Repository). BOR-based business objects offer a BDE interface; they do not support the use of BODs. The BOR can still be used in Enterprise Server 8.3 and higher to maintain existing business objects. Also Integration Studio 6.1 (which is used with Enterprise Server 8.3 or higher) does not support the use of BODs.

Background Information

For more information see:

Online help for the LN Studio and LN Implementation Generator.

Chapter 2: Creating a Business Interface Implementation for LN

Basically, the process for modeling business interface implementations is the same for any application server.

For example, you can start from a specified business interface definition (BID), import the required table definitions from the application server, define the mapping in the business interface implementation (BII), and generate the implementation for the application server.

Or you can import the required table definitions from the application server, generate a default BID and BII, adapt the BID and BII to your needs, and generate the implementation for the application server.

This chapter mainly contains information that is specific for LN.

Products and Compatibility

Overview

Until Enterprise Server 8.2, the Business Object Repository (BOR) was used to develop business objects for LN. From 8.3 onwards, the Integration Studio or LN Studio is used instead. The BOR is available in 8.3 and higher releases to allow maintenance on business objects developed in previous versions or feature packs. Only synchronous methods can be used for BOR-based business objects.

It is not advised to use the BOR for developing new business objects. Business object definitions can partially be migrated to the LN Studio through the Import from BOR feature, see Import from BOR on page 128.

Studio-based business objects can be developed and deployed as follows:

	Studio	Enterprise Server	Runtime
Synchronous and asynchronous BDEs	Integration Studio 6.1	8.3 or higher	Integration 6.2 ('OpenWorld' Adapter for LN)
	LN Studio 10.4	8.4.2 or higher	

	Studio	Enterprise Server	Runtime
Asynchronous BOD messages (See Implementing OAGIS BODs for LN on page 132)	LN Studio 10.4	8.4.2 or higher	ION

Focusing only on development, the compatibility is as follows:

	Enterprise Server 8.2 or lower	Enterprise Server 8.3, 8.4 or 8.4.1	Enterprise Server 8.4.2 or higher
Integration Studio 6.1	Unsupported	Supported	Supported
LN Studio 10.4	Unsupported	Unsupported	Supported

Important

The matrices mentioned before are independent of the LN application release (feature pack) that is used. For example, Enterprise Server 8.4.2 can be used in combination with the FP3 applications, which allows you to use either Integration Studio 6.1 or LN Studio. However, extensions on FP3 that are created using the LN Studio can only be deployed on environments using Enterprise Server 8.4.2. When developing a generic integration solution that must be available for all FP3 sites, use Integration Studio 6.1 instead, because those sites can use Enterprise Server 8.3.

Additionally, WSDL is only available in LN for business objects that were generated from the LN Studio. This WSDL is used in the LN Connector for Web Services.

Licensing and Product IDs

The LN Studio client is not licensed. Instead, the licensing is checked on the LN application server.

Basically, two product IDs are used:

- For development an LN Studio license (10146) is needed. Additionally, a development license for LN is needed.
- For using business objects at runtime, an Adapter license (7056) is needed.

Development Licensing

When using the Business Object repository (BOR) the following licenses are needed:

User Task	Required Product ID(s)	Notes
Modeling business objects in the BOR (session ttadv7500m000)	7033 (Business Data Entity Modeler) or 7105 (LN Studio) or 10146 (LN Studio)	A development license is needed.

User Task	Required Product ID(s)	Notes
Generating a BOL from the BOR (using the 'Convert BOR to runtime' command)	7034 (Business Data Entity Implementation Generator for LN) or 7105 (LN Studio) or 10146 (LN Studio)	A development license is needed.

Note: In Enterprise Server 8.3 or higher, only use the BOR for maintaining existing business objects. New business objects must be developed using the LN Studio.

When using the LN Studio the following licenses are needed:

User Task	Required Product ID(s)	Notes
• Modeling a BID, BII etc. in the LN Studio • Generating client proxies or WSDL • Import BID from LN • Import WSDL from LN	None	
• Table Importer • Form Importer • Module Importer • Import from BOR • Import a BI from LN	7105 (LN Studio) or 10146 (LN Studio)	A development license is needed. For the import from BOR also the availability of the package's sources is required. Also when importing a BI the user must be authorized for LN source code. Additionally the user must be authorized for Tools if the business object is in a Tools package.
Generating an implementation from the LN Studio into LN	7105 (LN Studio) or 10146 (LN Studio)	

Runtime Licensing

At runtime, the license checks are the same for all business objects, independent of their origin (BOR, Integration Studio 6.1 or LN Studio).

User Task	Required Product ID(s)	Notes
Using a BDE or BOD business object at runtime	7013 (Open Architecture Adapter 2.6 for LN) or 7056 (Open Architecture Adapter 2.7 for LN, Integration 6.2, ION LN Adapter)	Adapter 2.7 does not work with Enterprise Server 8.1 or lower

User Task	Required Product ID(s)	Notes
Retrieving meta data from the BOR at runtime	7035 (Business Data Entity Repository)	Only relevant for BOR-based business objects, when using the BOI Studio to create so called 'BOL-based BOIs' (through 'Import from BOR')

Using the Table Importer

Preparations

To use the Table Importer, you must have development authorizations. You can only import tables if you can view them in the **Table Definitions (ttadv4520m000)** session.

To use the Table Importer, a connection to the runtime repository is required:

- If you use LN Studio with an Infor LN 10.4 server, related software projects can be defined in the LN Studio preferences. If a related software project is defined for your interface project, the importer uses the runtime address of the software project.

  For details about related software projects, see these topics:

  - "Defining a default set of related software projects" in the LN Studio online help and the *Infor LN Studio Administration Guide*.
  - "Creating an interface project" in the LN Studio online help.
- If no related software projects are defined, you must specify the connection to the runtime repository. For details, see "Defining connectivity settings" in the *Infor LN Studio Administration Guide*.

Finally, an interface project must be available or created to import the table(s) into. See "Creating an interface project" in the LN Studio online help.

Importing Tables

1. Right-click the project that must contain the imported definitions and select **Import**.
2. Select the **Table** option and click **Next**.

   A list of available tables in the LN environment will appear if the connection to the LN environment is set up correctly (as explained earlier). The list is based on the user's package combination. You will not see tables from packages or package versions that are not included in the package combination.
3. Select the table(s) you want to import.
4. Click **Finish** to start the import.

Modeling the Business Interface Implementation

Overview

A number of things must be taken into account when creating a BII for LN. The LN Implementation Generator will check many constraints automatically, but not all. The most important constraints are described in the following section.

Component and Attribute Implementations

Identifiers

If a component is mapped to a table, then the identifying attributes for that component must be mapped to the identifying columns of the table.

Aggregated Components

If the parent or child component is mapped to a table then:

- Unrelated attribute implementations must be defined for each identifying attribute that is not linked to the component's identifiers but is related to the identifier of a parent component.
- An aggregation relation must be defined. The relationship implementation attributes in that aggregation relation must relate to identifying columns of the parent and child. Each of the parent's identifiers must be used in that aggregation relation.
- Calculated attributes must not be used to define an aggregation relation. Relations between components must always be defined at the 'inside'. For example, a public identifying attribute is used for a component which is mapped to two internal identifiers in the component implementation. In that case, the internal identifiers must be used in the aggregation relationship to child component implementations.

Duplicate Component Names

In the LN Studio a 'Public Name' element is available for components in a BID. If filled, this will be used as the name in the public interface (tag in request and response XML). If the public name is empty or missing, the component name will be used for that.

The Component.name (including BusinessInterfaceDefinition.name) identifies the component. It will always be filled for a component. In LN it is used for generating variable and function names, including

those used in the protected interface. But in the public interface, the public name will always be used if available.

From an LN point of view, the use of duplicate component names is possible for both BODs and BDEs. However, the proxies (Java or .Net) as generated for the BDEs may or may not support this.

Non-Root Tables

When using a non-root table, the preferred situation is having the identifier of the non-root table match the identifier of the root table. In that case there is a one-to-one aggregation relation between the tables. However, more complex situations are supported, as described in Modeling BIIs for Complex Cases on page 111.

Data Types

Every attribute implementation must have a data type specified, except when it is linked directly to a table column or implemented through an association. A data type that is used in a BII must always have a native data type that contains an existing LN domain.

If standard conversion is applicable (UTC, boolean, enumerate, or text) then the public data type must match that conversion, unless a conversion hook is specified. If no standard conversion is applicable then public data type must match the native data type (domain), unless a conversion hook is specified.

Data Types having Dimensions

In the LN Studio, repeatable attributes are modeled by setting the ‘dimensions’ for a data type. Repeatable attributes are attributes that can occur multiple times within their parent. This concept is comparable to array columns as used in the LN data model.

In a business interface implementation, the following can be modeled for LN:

- One-to-one mapping:

  Use an attribute having n dimensions, use an array column having the same number dimensions, use an attribute implementation having the same number of dimensions. In theory an attribute (and attribute implementation) can have more dimensions if it is input-only. And an attribute (and attribute implementation) can have fewer dimensions if it is output-only. But these situations are unsupported in a one-to-one mapping. In such cases a calculation (see below) must be used instead.

- Element mapping:

  Use an array column (data type of table column has multiple dimensions) and map one or more elements to attribute implementations. This is done through the ‘Additional Instructions’ property. For example, a table has a column prcs (Prices). Three attributes are defined, being costPrice, localPrice and cataloguePrice. These attributes must be mapped to the first three array elements of the prcs column. In that case, map costPrice to column prcs and in the Additional Instruction for the attribute implementation, fill in “1”. For localPrice fill in “2” and for cataloguePrice use “3”. The

attributes and attribute implementations use a data type with no dimensions. Only the table column will use dimensions.

- Using a calculation:
  - On get hook, calculated attribute implementation has dimensions, and used attributes have no dimensions.
  - On get hook, calculated attribute implementation has dimensions, and used attributes have one or more dimensions.
  - On get hook, calculated attribute implementation has no dimensions, and one or more used attributes have multiple dimensions
  - On set hook, attribute implementation has multiple dimensions, and used attributes have no dimensions.
  - On get hook, attribute implementation has multiple dimensions, and used attributes have one or more dimensions.
  - On set hook, attribute implementation has no dimensions, and one or more used attributes have multiple dimensions

It is not possible to map a repeatable attribute directly to separate rows of a table for the component. For example, component Order maps to the Order table. Attribute Order.note having multiple dimensions maps to the rows of an OrderNotes table. In such case, the note attribute implementation must be implemented using an on get and on set hook, which reads and updates the table (through its DAL).

The same domain can be used in LN for table fields with or without dimensions. In that case multiple datatypes are needed, one for every combination of domain (native data type) and dimensions value. Each data type must have a unique name, but the native data type will then be the same. For example:

	Datatype without dimensions	Datatype having 3 dimensions
Datatype	ppmmmvalu (or any other name)	ppmmmvalu3 (or any other name)
Native data type	ppmmmvalu	ppmmmvalu
Dimensions	empty	3

When using repeatable attributes, the constraint are applicable for LN:

- In case of a ‘one-to-one mapping’, the value of the dimensions property must match.
- In case of an ‘element mapping’, the specified element must exist. In other words, the element must be greater than 0 and less than or equal to the number of elements for the array column.
- If filled, the dimension must always contain a number. Other values (including for example multi-dimensional arrays such as “10, 20”) are not allowed.
- A dimension can only be used for a leaf attribute. So it cannot be used for a ‘complex’ data type or for an attribute group.
- A qualifier cannot be repeatable.

A repeatable attribute can be defined as dynamic by setting the dimension to 0. In that case the number of repeatable elements does not need to be known at beforehand. This is useful for processing incoming BODs.

Developing Hooks

The use of dimensions impacts the interface for hooks.

For set/get hooks, the interface is for example, an on set hook defined for an attribute implementation 'note' having dimension 5. Three used attributes are specified. The first attribute 'optionalNote' has no dimension and the other ones ('extraNotes' and 'numbers') have dimension 2. The last one ('numbers') is a long array, while the other ones are strings.

In that case the hook content for the on set hook is for example:

```
if i.optionalNote.iSet then
  o.note(1,1) = i.optionalNote
 else
  io.note.isSet(1) = false
 endif
 o.note(1,2) = i.extraNotes(1,1)
 o.note(1,3) = i.extraNotes(1,2)
 o.note(1,4) = str$(i.numbers(1))
 o.note(1,5) = str$(i.numbers(2))
 return(0) | OK
```

Note that strings have an extra dimension.

For method hooks (before/on/after get/set), the interface is comparable.

This is an example of a before execute hook:

```
if not i.code.isSet(1) or not i.code.isSet(2) then
  io.cancel = true
 endif
 return(0) | OK
```

This is an example of an on execute method hook:

```
 #pragma used dll oppmmmdll0001
 return( ppmmmdll0001.calculate.quantites.and.add.notes(i.item,
i.item.isSet,
         i.description(1,1), i.description.isSet(1),
         i.description(1,2), i.description.isSet(2),
         io.quantity(1), io.quantity.isSet(1),
         io.quantity(2), io.quantity.isSet(2),
         o.note(1,1), io.note.isSet(1),
         o.note(1,2), io.note.isSet(2)) )
```

Protected Interface for Repeatable Attributes

In LN, the protected interface is implemented using setters and getters instead of function parameters.

In case of dimensions, arrays are not used, because then you cannot set or get individual elements. If element n is set, the other elements must not be set. Instead, an additional parameter is used for the element number. However, internally arrays are used for the corresponding ‘bl’ variables.

For a ‘normal’ attribute implementation, a setter function has this interface:

```
function extern
long ppmmm.bl999st00.set.MyComponent.myAttribute(
  const domain ppmmm.str1 i.value
```

For an array attribute implementation, the setter function will have this interface:

```
function extern
long ppmmm.bl999st00.set.MyComponent.myAttribute(
    long    i.element,
  const domain ppmmm.str1 i.value)
```

In the same way, the getter function will have this interface:

```
function extern
long ppmmm.bl999st00.get.MyComponent.myAttribute(
    long    i.element,
  ref  domain ppmmm.str1 o.value)
```

Protected Associations and Protected Methods

You can model association relationships in the BII, and you can model methods in the BII for which no corresponding method exists in the BID. See the LN Studio online help for more information.

When generating an LN implementation, protected methods will not only get a protected interface in the generated 'st' library, but also an XML-based interface in the generated 'sb' library. You can forward a request XML from a public method to a protected method

Using checksums in BDEs to prevent data corruption

In integrations, BDEs are used for synchronizing data between the outside world and LN. When modifying data using BDEs, the data is not locked as in a regular LN transaction. Between two subsequent BDE calls (for example a Show to retrieve the data and a Change to modify), the business object data can be modified within LN without invoking the BDE. This can result into data corruption.

A mechanism to prevent this data corruption is to apply checksum checking. The BDE is extended with a checksum attribute for each component. When a Change request is performed, it will be checked if the business data has left the same in the meantime by calculating the checksum. The checksum can be determined by performing the Show method.

If the checksum differs, the integration application can report "`Record changed by other user`" and perform a new Show to refresh the data. The user can retry then.

Code generation

For each component having a Change method, two functions are generated that calculate the checksum.

```
function string <ComponentName>.CalculateChecksum()
```

Returns the checksum as a single-byte string of length 40.

The checksum is based on all table fields linked to an attribute implementation of all tables that are used within the component or one of its ancestors. These table fields are concatenated into one single string and passed to function calculate.checksum() that uses the Portingset SHA functions to calculate a checksum.

```
function string <ComponentName>.CalculateChecksumRB()
```

Returns the checksum as a single-byte string of length 40.

The checksum is based on the complete record buffer of all tables that are used within the component or one of its ancestors. These table buffers are encoded and use the Portingset SHA functions to calculate a checksum.

Using the Checksum check

In the model: model a public calculated attribute for the checksum for each relevant component and implement it in the this way:

Datatype: a string with length 40.

In the OnGet hook: invoke the CalculateChecksum method for the component and assign it to the checksum attribute, for example:

```
o.checksum = Items.CalculateChecksum()
```

In the BeforeExecute hook: check the old value of the checksum (provided via the change request) and return an error if the values do not match. For example:

```
if i.checksum.isSet and (strip$(i.checksum) <> Items.CalculateChecksum())
 then
        dal.set.error.message("@Instance changed by other user!")
        io.cancel = true
endif

return(0)
```

Note: If the provided checksum calculations are not sufficient (for example, a calculation based on a DLL call may be involved or some particular fields may be sufficient), you can write your own checksum functions based on the functions that are generated by LN Studio and call calculate.checksum() to calculate the actual checksum as 40 character single-byte string.

Customizations in LN Business Interfaces

Standard business interfaces for LN (both BODs and BDEs) can be customized by including custom data (either from customer-defined fields or from 'normal' customizations) in a standard business interface.

The LN Studio model is extended, to enable the developer of standard business interfaces to define where customized data must be included. For example, select a 'UserArea' element for each business interface definition component.

The customer can create a 'customization library' specifying the columns that must be included in the business interface.

The solution is limited to simply including data for custom table columns (including customer-defined fields) in both incoming and outgoing messages. The data is always included in the designated elements (such as UserArea elements). The customer cannot use other existing but unused BOD elements. Also table columns that have a reference to other tables cannot be used.

Customer-defined fields and normal columns are added to an existing business interface. The custom data is handled in the List, Show, Create and Change methods. The standard business interface can be changed and regenerated without impacting the customization.

Using custom data in a Business Object

When including custom data in an existing business object (either BDE or BOD), use a customization library.

With a customization library you can extend the business object to handle:

- data from table columns that are standard inLN , but not used in the BDE/BOD
- data from table columns that are created as a customization;
- data from so-called 'customer-defined fields' (CDF), which are defined through the Customer-Defined Fields (ttadv4194m000) session.

You can create customizations without updating or merging them when a new version of the standard is installed. A new version of a BDE or BOD can be delivered, for example in a solution or a new feature pack. Installing such a new version does not overwrite the customization. In fact, the customization usually does not have to be changed at all.

Exceptional situations in which the customization must be changed:

- An interface change in the BDE/BOD that breaks the compatibility, such as removing a component.
- Changing the implementation of a BDE/BOD in such a way that the table containing the custom data is not used anymore by the implementation.

Additionally, if the customization element is changed for a component then the customization library will still work, but the customization data will be included in the new location instead of the old location. This will impact the other application that communicates with LN through BDEs or BODs.

Creating a customization library

To create a customization library:

1. Create a new customization library in your customization VRC.

   The customization library code is comparable to the two generated libraries, except that it uses 'cc' instead of 'sb' or 'st'. For example, if your implementation identifier is ppmmm123 then the customization library will be ppmmmbl123cc00. A customization library can only be created in a customization VRC. It will never be a part of the standard.

2. Copy the template into the new library.

   For example, if your implementation identifier is ppmmm123, then open the ppmmmbl123sb00 library. Copy the lines between (but excluding) "#ifdef CUSTOMIZATION_LIBRARY" and "#endif | CUSTOMIZATION_LIBRARY" to your new ppmmmbl123cc00 library.

3. For each component that requires custom data, specify the required fields using the addTableField() and addCdfField() methods.

   The parameters to be used in addTableField() and addCdfField():

   - **`elementName (string):`** the name that must be used to identify the element in the business object data XML.
   - **`tableCode (string)`**: the code of the table that contains the custom data to be included.
   - **`columnName or cdfName (string)`**: the name of the table column or customer-defined field. Note that customer-defined fields have a prefix ('cdf_') in the table column name. Do not include this prefix in the cdfName.
   - **`dataType (string)`**: the data type of the column or customer-defined field.

   These data types can be used:

   - String
   - Integer
   - Numeric
   - Date (for UTC date/time)
   - DateOnly (for Date without time)
   - Checkbox

At Runtime

The custom data is included in a predefined way in the custom data element.

For example, if a customization library contains:

```
case "Order":
 | Custom data element for this component is Order.Header.UserArea.
 | Fill in your table fields here.
 | You can use data from one of the following table(s):
 | ppmmm123.
 addTableField("CustomDate", "ppmmm123", "cdat", "Date")
 addCdfField("SomeNotes", "ppmmm123", "cstr", "String")
 addCdfField("CustomNumber", "ppmmm123", "cnum", "Integer")
 addTableField("MyFlag", "ppmmm123", "flag", "Checkbox")
```

```
 addTableField("MyAmount", "ppmmm123", "amnt", "Numeric")
 break
```

Then the XML as used to contain the BOD/BDE data will contain for example:

```
<UserArea>
 <Property>
  <NameValue name="ln.cust.CustomDate" type="DateTimeType">2009-06-
24T14:00:00Z</NameValue>
 </Property>
 <Property>
  <NameValue name="ln.cust.SomeNotes" type="StringType">Customer-specific
 order notes</NameValue>
 </Property>
 <Property>
  <NameValue name="ln.cust.CustomNumber" type="IntegerNumeric
Type">10</NameValue>
 </Property>
 <Property>
  <NameValue name="ln.cust.MyFlag" type="IndicatorType">true</NameValue>
 </Property>
 <Property>
  <NameValue name="ln.cust.MyAmount" type="NumericType">25.95</NameValue>

 </Property>
</UserArea>
```

The custom data is handled automatically for:

- BDE List/Show response.
- Events or BODs that are published using one of the ShowAndPublish methods.
- Standard BDE Create and Change methods, and for incoming BODs that are handled through these methods.

The custom data is not handled for other methods. It is also not handled for List, Show, Create and Change methods that have a specific batch implementation through an on execute hook.

Custom data will not be handled if the corresponding database row does not exist. This can be the case if a component is mapped to two tables, while a row is not mandatory for the secondary table.

Prerequisites

The BOD/BDE component must have an element marked as the location for custom data. This element can have any name (for BODs it will usually be 'UserArea'), and it must be either an anyType or an attribute group.

Only one custom data element can be used per component. The generated customization library template indicates for which components a custom data location is defined. If no component has a custom data location defined, no customization library template will be generated.

Only columns of tables that are used in the BII for the component can be used.

The custom data must not be in a text or blob column.

Customization Library Template

Template code for the cc library is generated in the sb library. The template contains assisting defines to invoke the tlbctinterface library, including error handling. Additionally, some (commented) example code is included to demonstrate how to use the customization library. All data types that can be used are documented.

For example, when generating the runtime for a ppmmm800 business object the template can contain:

```
|#############################################################################
|#################### Customization Library part
################################
|#############################################################################

| Customization library for the Order business object.
| Do not use compile flag CUSTOMIZATION_LIBRARY. This flag is used to es
cape
| the code below, because it is not a part of the ppmmmbl800sb00 library.
| You copy the code below to the ppmmmbl800cc00 in your customization VRC
| and use it as a starting point for a customization library.

#ifdef CUSTOMIZATION_LIBRARY

|------------------- DO NOT CHANGE THIS PART --------------------------
-------

| Template for ppmmmbl800cc00 (Order)
| Template generation date: 26-Feb-2009 16:45:25

#pragma used dll "otlbctinterface"

#define initialize()
^ retl = tlbctinterface.cust.initialize(o.xml)
^ if retl <> 0 then
^  return(retl)
^ endif

#define addTableField(elementName, tableCode, columnName, dataType)
^ retl = tlbctinterface.cust.add.element(o.xml, elementName,
^     tableCode, columnName, dataType)
^ if retl <> 0 then
^  return(retl)
^ endif

#define addCdfField(elementName, tableCode, cdfName, dataType)
^ addTableField(elementName, tableCode, "cdf_" & cdfName, dataType)

| Use addTableField to include data for a table field as defined in the
| data model. This could be a customer-specific table field, or a table
| field that exists in the standard, but is not yet used in the business
 object.
| Use addCdfField to include data from a customer-defined field in the
| business object. Customer-defined fields are maintained in the
| Customer-Defined Fields (ttadv4194m000.) session.
|
```

```
| The parameters to be used are:
| 1. elementName (string): the name that must be used to identify the
|    element in the business object data XML.
| 2. tableCode (string): the code of the table that contains the custom
|    data to be included.
| 3. columnName or cdfName (string): the name of the table column or
|    customer-defined field. Note that customer-defined fields have a
|    prefix ('cdf_') in the table column name; this prefix must NOT be
|    included in the cdfName.
| 4. dataType (string): the data type of the column or customer-defined
|    field.
|    The following data types can be used:
|    - "String"
|    - "Integer"
|    - "Numeric"
|    - "Date"
|    - "DateOnly"
|    - "Checkbox"
|
| Some examples:
| addTableField("CustomDate", "ppmmm001", "cdat", "Date")
| addCdfField("CustomNote", "ppmmm001", "note", "String")
| addTableField("CustomAmount", "ppmmm001", "camn", "Numeric")
| addCdfField("NumberOfDeviations", "ppmmm001", "cdev", "Integer")
| addTableField("Checked", "ppmmm001", "cchk", "Checkbox")

|------------------- Custom implementation ----------------------------
-------

function extern long ppmmm.bl800cc00.get.additional.elements(
 const string i.component.path,
 ref long o.xml)
{
DLLUSAGE
Desc Define the additional elements (columns) for a component.
Input i.component.path - the component path.
Output return value - 0 (OK) or DALHOOKERROR (one or more DAL error
  messages are set)
 o.xml - if return value is = 0: an xml structure containing
  the details on custom fields (if any)
ENDDLLUSAGE

 long retl  | return value to be checked

 initialize() | do not remove this line

 on case i.component.path

 case "Order":
  | Custom data element for this component is Order.Header.UserArea.
  | Fill in your table fields here.
  | You can use data from one of the following table(s):
  | ppmmm800.
  | For example:
  |addTableField("CustomDate", "ppmmm800", "cdat", "Date")
  |addCdfField("AdditionalNote", "ppmmm800", "note", "String")
```

```
  break

 case "Order.OrderLine":
  | Custom data element for this component is Order.OrderLine.UserArea.
  | Fill in your customized table fields here.
  | You can use data from one of the following table(s):
  | ppmmm801, ppmmm802.
  | Custom data from the ppmmm802 table will only be used when a
  | corresponding row exists in that table.
  | For example:
  |addTableField("CustomDate", "ppmmm801", "cdat", "Date")
  |addCdfField("AdditionalNote", "ppmmm801", "note", "String")

  break

 | no custom data location available for component "Order.OrderLine.Sub
Line"

 default:
  | Nothing
 endcase

 return(0) | OK
}
```

Feature Pack Independent BIIs

BODs and BDEs for LN are often implemented for multiple LN feature packs, such as FP6, FP7, etc. Implementations of the same interface can be created for Baan IV and Baan 5.0

Solution in the Generated Code

In case of data model differences between the feature packs or versions, multiple business interface implementations (BIIs) can be created in the LN Studio. However, this requires additional development efforts and makes maintenance more expensive. For that reason it is advisable to create a single BII and use that to generate the runtime in multiple feature packs or versions. Or generate the BII in a separate package and use a single version of that package in combination with multiple application versions.

However, in new application versions new table columns may be introduced. If those table columns are used in the BII, the BII cannot be generated for the older versions, in which the table columns are missing. For that reason the LN Studio generator is enhanced. The generated code has become fault tolerant for missing table columns.

If columns a, b and c exist in FP2 and FP5, but column d only exists in FP5, you can use a single implementation for both feature packs. The generated code will ensure that for FP2 column d is not selected or assigned, but for FP5 it is.

Behavior at Runtime

What happens when a table column is used in a BII, but that column is unavailable in the application version that is used at runtime.

In case of outgoing data (such as a BDE Show response or a BOD being published):

- An attribute implementation that is linked directly to a missing column will not be selected and will not be set.
- If an attribute implementation is mapped to a missing column and that attribute implementation has a default value defined, that default value is used if the column does not exist at runtime. (Note that a default value can only be used if the attribute implementation data type does not have multiple dimensions.)

In case of incoming data (such as a received BOD or a BDE request):

- No value is assigned to the missing column. The incoming data is ignored.
- The fault tolerance deals with the situation where a column is an array (the data type has multiple dimensions) and the column exists at runtime but it has less elements than defined in LN Studio. In that case, the additional elements are ignored, both for outgoing and incoming data.
- When filtering (in a List, Show, SubscribeList or SubscribeEvent request) on an attribute that is linked to a non-existing column, the filter is handled as a postfilter. In postfiltering, missing values are regarded as empty. A filter on Attribute1 <> "ABCD" will evaluate to true, while a filter on Attribute2 > 10 will evaluate to false. Filters on attributes that are indirectly derived from non-existing columns though 'on get' hooks are also handled as a postfilter (just like any filter on a calculated attribute).

Development Guidelines

The handling of missing columns is done automatically by the generated code. You do not have to adapt the BII for this.

Note the following when developing a BII for multiple application versions or feature packs:

1. Because of the fault tolerance, errors for non-existing columns that should exist will not be detected when generating anymore.
2. The solution will also work for Baan IV and Baan 5.0. Even though the data model differences between Baan IV, Baan 5.0 and LN will probably not be limited to new columns in the newer versions.
3. Attribute implementations that are linked to missing columns will not be set when producing outgoing data. If such attribute implementations are used in hooks, ensure the hook implementation can handle that. You can use the 'isSet' boolean to check whether a value is available. For example:

```
if i.attribute2.isSet then
 o.calculatedAttribute = i.attribute1 & i.attribute2
else
 o.calculatedAttribute = i.attribute1
endif
```

4. For incoming data, attribute implementations are handled as usual even if they are linked to a non-existing column, so they will be set based on the incoming data. They can be used in 'on set' hooks for another attribute implementation, for example.
5. Usually the latest data model will be the most complete one. In case of localizations there can be situations where no complete data model exists at any back-end. For example, a and b exist in all versions, c exists in FP5, d does not exist in any standard FP, but does exist in a localization based on FP2. In that case the additional column d from the localization has to be merged into the generic data model in the LN Studio. In other words, the table in LN Studio must be the union of all versions for which the implementation is used.
6. Check the limitations as described in the next section.

Limitations

The limitations:

1. Only missing table columns are handled in a fault-tolerant way. Missing tables are not.
2. Fault-tolerant handling is not used in these situations:
   a. When a column is part of index 1.
   b. When a column linked to an identifying attribute of component.
   c. When a column is used in component implementation relationship.
   d. When a column is used in component implementation table relationship (join of a non-root table).
   e. For customized fields (see Customizations in LN Business Interfaces).
3. Attribute implementations must use a domain that exists in all application versions for which the BII is used. If not only the column does not exist in some versions, but also the domain of that column does not exist, the attribute implementation must use another domain. It is no problem to use a non-existing domain in the table artifact, as long as the attribute implementations use a domain that exists in all application versions for which the BII is used.

   Note that this will result in a warning when generating the implementation

   ```
   (LN constraint: Attribute implementation
   MyObject.MyAttributeImplementation should not have a data type specified,
   because it is mapped directly to a table column
   ```

   As long as you ensure the used data types (domains) in attribute implementation and column are compatible you can ignore this warning.
4. Function server implementations are not taken into account. Note that they can have issues that are comparable to tables: in new releases new form fields can be added.
5. The solution does not cover filter hooks.

Filter Hooks

If a field that does not exist in all feature packs is used in a filter hook, you can deal with that as follows.

Library function:

```
function extern boolean tcbod.dll0001.table.field.exists(
       const string i.table.field )
{
DllUsage
```

```
Expl This function checks if a specified table field exists.
Input i.table.field: table field code including table, e.g. "ppmmm999.abcd"
Output n.a.
Return true  - table field exists.
 false - table field does not exist.
EndDllusage

 long   return.value
 long   dummy.position.field.in.row
 long   dummy.size.field
long   dummy.depth.field
 long   dummy.type.field
 long   dummy.flags
 string dummy.domain.name
 string dummy.default.value

 return.value = rdi.column(
                     i.table.field,
                     dummy.domain.name,
                     dummy.position.field.in.row,
                     dummy.size.field,
                     dummy.depth.field,
                     dummy.type.field,
                     dummy.flags,
                     dummy.default.value )
 return(return.value = 0)
}
```

Define in include hook:

```
#define PPMMM999_FLD2 "ppmmm999.fld2"
 | define is needed for filter hook, because otherwise the table code is
 replaced if
an alias is used
```

Filter hook

```
o.hookFilter = "ppmmm999.fld1 >= 0"
if tcbod.dll0001.table.field.exists(PPMMM999_FLD2) then
 | field fld2 may not exist in all feature packs, so we must check this
 o.hookFilter = o.hookFilter & " and ppmmm999.fld2 = tcyesno.no"
endif
return(0)
```

Solution in the BII/Application

For complex data model differences (such as missing tables), a feature pack-independent BII can be created by handling the differences in hooks.

In general, you can use this approach:

1 To reduce the efforts for implementing and maintaining BODs for LN feature packs, reuse as much as possible. Create a specific implementation only for those parts of the BII that require a specific

implementation. Avoid duplicates of specific code. Do not copy a lot of code to adapt only a minor part.

2 Use a single BID and BII for all feature packs. Generate the runtime in a separate package having a single version independent of the LN FP (but multiple versions for new BOD versions). Implement application version-specific code in the application package version.

The specific DLLs in each feature pack will offer the same interface; only the implementation will be different.

For any difference in data model (or used application DLL functions), a solution is already possible using the current LN Studio and LN implementation generator.

Use an 'on get hook' for the field(s) that are not applicable in an older application version. In the hook, invoke a function from an application DLL. Implement that function differently depending on the application version.

For example in FP2

```
o.attribute = ""
o.attribute.isSet = false
return(0)
```

In FP5

```
select table.column:o.attribute
from table
where table.key = :i.key.attribute
selectdo
 o.attribute.isSet = true
selectempty
 ...
selecterror
 ...
endselect
return(0)
```

If multiple columns are involved, an optimization can be implemented combining the selects for the same table.

Pro: on BII for all LN FPs. No technology changes needed.

Con: an additional query is executed on the same table.

To be determined:

- Using 'alternative implementations' may also help to reuse BII parts that are equal?
- What about BID changes other than the addition of attributes, such as introduction of a new component?

Example:

In Baan IV, two columns exist (ppmmm123.date and ppmmm123.time). In LN , one column exists (ppmmm123.utcd).

Assume the following generic functions exist in both versions:

- tcabc.dll0001.date.and.time.to.utc(long i.date, long i.time, ref long o.utc)
- tcabc.dll0001.utc.to.date.and.time(long i.utc, ref long i.date, ref long i.time)

To deal with this in a version-independent manner, ensure the ppmmm123 table in LN Studio contains three columns: date, time and utcd.

Use these attribute implementations:

- date: maps to ppmmm123.date, uses an on set hook based on DateTime
- time: maps to ppmmm123.time, uses an on set hook based on DateTime
- utcd: maps to ppmmm123.utcd, uses an on set hook based on DateTime
- DateTime: calculated using an on get hook based on date, time and utcd.

On set date:

```
long dummy.time
tcabc.dll0001.utc.to.date.and.time(i.DateTime, o.date, dummy.time)
return(0) | OK
```

On set time:

```
long dummy.date
tcabc.dll0001.utc.to.date.and.time(i.DateTime, dummy.date, o.time)
return(0) | OK
```

On set utcd

```
o.utcd = i.DateTime
return(0) | OK
```

On get DateTime

```
if i.utcd.isSet then
 o.DateTime = i.utcd
else
 tcabc.dll0001.date.and.time.to.utc(i.date, i.time, o.DateTime)
endif
return(0) | OK
```

Alternatively you can delegate the logic to a version-dependent library function. For this example it can be a bit overdone, but in more complex situations this approach can be helpful. In that case the code is:

```
ppmmm.dll0001.get.date.from.utc.if.applicable(i.DateTime, o.date,
io.date.isSet)
return(0) | OK
```

On set date:

```
ppmmm.dll0001.get.date.from.utc.if.applicable(i.DateTime, o.date,
io.date.isSet)
return(0) | OK
```

On set time:

```
ppmmm.dll0001.get.time.from.utc.if.applicable(i.DateTime, o.time,
io.time.isSet)
return(0) | OK
```

On set utcd

```
ppmmm.dll0001.get.utc.from.utc.if.applicable(i.DateTime, o.utcd,
io.utcd.isSet)
return(0) | OK
```

On get DateTime

```
ppmmm.dll0001.get.utc.from.utc.or.date.and.time(i.date, i.time, i.utcd,
 o.DateTime, io.DateTime.isSet)
return(0) | OK
```

Library ppmmmdll0001 in Baan IV

```
function extern ppmmm.dll0001.get.date.from.utc.if.applicable(
  long i.utc,
 ref long o.date,
 ref long io.date.isSet)
{
 long dummy.time
 tcabc.dll0001.utc.to.date.and.time(i.utc, o.date, dummy.time)
}

function extern ppmmm.dll0001.get.time.from.utc.if.applicable(
  long i.utc,
 ref long o.time,
 ref long io.time.isSet)
{
 long dummy.date
 tcabc.dll0001.utc.to.date.and.time(i.utc, dummy.date, o.time)
}

ppmmm.dll0001.get.utc.from.utc.if.applicable(
  long i.utc,
 ref long o.utc,
 ref long io.utc.isSet)
{
 io.utc.isSet = false
}

ppmmm.dll0001.get.utc.from.utc.or.date.and.time(
  long i.date,
  long i.time,
  long i.utc,
 ref long o.utc,
 ref long io.utc.isSet)
{
```

```
 tcabc.dll0001.date.and.time.to.utc(i.date, i.time, o.utc)
 | i.utc is unused in Baan IV
}
Library ppmmmdll0001 in ERP LN, same interface but a different imple
mentation:

function extern ppmmm.dll0001.get.date.from.utc.if.applicable(
  long i.utc,
 ref long o.date,
 ref long io.date.isSet)
{
 io.date.isSet = false
}

function extern ppmmm.dll0001.get.time.from.utc.if.applicable(
  long i.utc,
 ref long o.time,
 ref long io.time.isSet)
{
 io.time.isSet = false
}

ppmmm.dll0001.get.utc.from.utc.if.applicable(
  long i.utc,
 ref long o.utc,
 ref long io.utc.isSet)
{
 o.utc = i.utc
}
ppmmm.dll0001.get.utc.from.utc.or.date.and.time(
  long i.date,
  long i.time,
  long i.utc,
 ref long o.utc,
 ref long io.utc.isSet)
{
 o.utc = i.utc
 | i.date and i.time are unused in ERP LN
}
```

Attribute Grouping

If one attribute from a group is mandatory then the group must also be mandatory. If an attribute group is mandatory it must contain at least one mandatory attribute. If an attribute group is read-only then all its children must be read-only.

Action Types

A standard Change method offers support for actionType qualifiers per component instance; this must not be modeled in the BID or BII. The use of actionType component qualifiers is also supported for PublishEvent and OnEvent. They cannot be used for standard methods other than Change, PublishEvent and OnEvent. For the standard Change method, only the following actionType component qualifier values are allowed: ‘create’, ‘change’, ‘delete’, ‘unchanged’ or ‘createOrChange’. The ‘createOrChange’ actionType is handled top-down, just like the ‘create’ and ‘change’ action types.

A specific method can have any actionType component qualifier; this will be handled just like any other qualifier attribute.

Methods and Method Arguments

Method Implementations per Component

Methods having processing scope ‘batch’ must only be implemented on the top-level component.

If a top-down or bottom-up method is implemented on a subcomponent, it must also be implemented on its parent(s).

Method Arguments

LN does not support method arguments having scope ‘unknown’, ‘result set’ or ‘return’.

Also controlling attributes cannot be used as method arguments for LN, except for the standard controlling attributes of standard methods, which are not explicitly defined in the LN Studio.

Method Argument Sequence

For each component, the sequence (position) of the identifying PublishEvent arguments must match sequence of the identifying Show arguments.

Method Argument Implementations

For each method implementation having arguments, each of the component’s identifiers must be available as a parameter (input, input/output or output, depending on the method).

Other Guidelines

BII Names

Usually, your BII must have the same name as the BID it corresponds to. If your business interface implementation is ‘protected’ or ‘private’ and has another name than the business interface definition, then the business interface implementation name is used as the business object name in LN. In that case your business object will be a protected business object, because it cannot be accessed from the outside through the interface as specified in the BID. For details, refer to section Using Multiple Implementations on page 120.

Hooks

Using Hooks for LN on page 46 describes how to implement hooks for LN.

Modules and Procedures

Methods can be implemented using an existing module/procedure. Import the module from LN and link a procedure it to a method implementation. Or invoke the procedure from a hook (refer to section Using Libraries and Functions in Hooks on page 46).

Traversal by Association

Traversal by association can be used. For details, refer to section Screen Scraping using the Application Function Server on page 111.

Tables of Type ‘Form’

Tables of type ‘form’ can be used; refer to Screen Scraping using the Application Function Server on page 111.

Name Lengths

Business object (BID or BII) names cannot be longer than 40 characters for LN. Other names cannot be longer than 50 characters.

Internal transaction handling in Business Objects

Transaction handling for BODs and BDEs is done as a whole. It is uncommon to have an internal transaction. For example, a transaction with an own retry point and commit to commit a part of the Business Object handling to the database regardless if the complete Business Object action will succeed or fail. There can be reasons to have an internal transaction within a BOD or BDE.

This can be established by using a subprocess.

Note: Before starting the subprocess, system variable *db.child.transaction* (set within the Dispatcher) must be temporarily reset by *db.set.child.transaction(0)*. This is to avoid suppressing the transactions that are performed within the subprocess. After starting the subprocess, *db.child.transaction* must be restored with *db.set.child.transaction(1)*. To avoid creating a subprocess for every invocation, it is advisable to reuse the subprocess and use BMS commumication between main process and subprocess.

Miscellaneous

Method-dependent guidelines on how to implement BDE methods for LN are listed in Method-Specific Guidelines on page 87. Guidelines for creating BIIs for complex situations are included in Modeling BIIs for Complex Cases on page 111.

Generating the Runtime

Preparations

BID and BII

First make sure that the BID and BII are correct. Solve any problems reported by the LN Studio. You may need to rebuild to view the actual list of problems (**Project > Build All**, or **Project > Build Project**).

Additionally, the BII must meet the guidelines for LN, as described earlier and in Modeling BIIs for Complex Cases on page 111.

Authorizations

The business object runtime can only be generated if you have got development authorizations. In other words, you can only create a business object runtime from the LN Studio if you are also able to create libraries in LN.

Connectivity

To use the generator, a connection to the runtime repository is required.

- If you use LN Studio with an Infor LN 10.4 server, related software projects can be defined in the LN Studio preferences. If a related software project is defined for your interface project, the generator uses the development address of the software project.

  For details about related software projects, see these topics:

  - "Defining a default set of related software projects" in the LN Studio online help and the *Infor LN Studio Administration Guide*.
  - "Creating an interface project" in the LN Studio online help.
- If no related software projects are defined, you must specify the connection to the runtime repository. For details, see "Defining connectivity settings" in the *Infor LN Studio Administration Guide*.

Implementation Identifier

The BII you want to generate must have an implementation identifier set. This code (for example ppmmm123) is a property of the BII. The code consists of eight characters:

- Two characters for the package code
- Three characters for the module code
- Three digits.

Once generated, the implementation identifier and business object name are inseparable. So the generation will fail if:

- The environment already contains another business object having the same implementation identifier.
- The environment already contains the same business object name having another implementation identifier. Also if that other business objects runtime is stored in another VRC.

To change the implementation identifier or name for an already existing business object, you must first delete the existing business object in any version where it exists, using the **Business Objects (ttadv7500m000)** session.

Version

Before generating, you must specify the version in which the business object runtime must be created. This is only required if no related software projects are defined.

Window > Preferences > Infor LN Studio Integration > Generators > Infor LN Implementation

The Major Version, Minor Version and (optional) Subordinate Version refer to what is called the Base VRC in LN. The available base VRCs can be viewed using the **Base VRCs (ttpmc0110m000)** session.

If no base VRC exists that corresponds to the VRC where you need to generate the business object runtime, a base VRC must be created in LN.

The actual package VRC that is used is determined as follows:

- The package is taken from the implementation identifier (mentioned earlier).
- The VRC is taken from the Export VRC as defined in the **Base VRCs (ttpmc0110m000)** session.

Note: You must have sufficient authorizations to develop in the resulting package VRC.

In Infor development, the VRC to be used is a VRC that is open for development (or maintenance). In a customer environment, the VRC is a customization VRC, such as B61C_a_cust.

Generating the Runtime

The business object implementation for LN is generated by right-clicking the BII and selecting **Implementation > Generate Infor LN Implementation**.

When generating, the LN Implementation Generator may report violations of LN-specific constraints. Constraint violations are likely to result in errors when generating or errors at runtime when invoking the business object in LN.

The following happens in LN:

- Two libraries are generated. The library code is derived from the BII's implementation identifier.
- A lookup entry is created, which is used at runtime to route the invocation of a business object method to the library implementing the public interface for that business object.

Note: If related software projects and activities are defined for your interface project, the libraries are generated in these activities. For details, see "Creating an interface project" in the LN Studio online help.

Compilation errors in the generated libraries are reported. Usually, these compilation errors will only occur for the hooks you included in your BII, or in case of constraint violations or problems reported by the LN Studio. However, not all problems are reported directly to the user, in some cases the compilation error is the only feedback you get on a problem.

Note:

If a BOR-based business object already exists in a previous version and you want to generate from the LN Studio, the following libraries must be duplicated to the new version and set on expired:

- sc
- sf
- sm

This is not done automatically! For example, if the implementation identifier is ppmmm999 then the following libraries will be regenerated in the new version:

- ppmmmbl999sb00
- ppmmmbl999st00

The following libraries will not be used anymore:

- ppmmmbl999sc00,
- ppmmmbl999sf00
- ppmmmbl999sm00

They must be set on expired in the new version.

Deleting or Expiring the Runtime

Deleting an Implementation

To remove a business object runtime, right-click the BII and select **Implementation > Delete Infor LN Implementation**. If related software projects are defined at the interface project, then the generated components are removed from the used activity.

If a business object was created in the **Business Objects (ttadv7500m000)** session in LN, then you can remove the business object runtime through that session. Additionally, you must then delete the libraries that were generated for the business object. The library code is derived from the implementation identifier (business object code). For example, for ppmmm999 these are the generated libraries:

- `ppmmmbl999sb00`
- `ppmmmbl999st00`

Note: The **Business Objects (ttadv7500m000)** session contains a read-only **Maintained in Studio** check box. If this check box is selected, then you cannot delete the business object in the **Business Objects (ttadv7500m000)** session. In that case, it must be deleted in LN Studio.

The meta data (BI file) is an ‘additional file’ that is linked to the lookup entry. This file is automatically deleted when deleting the lookup entry from the Business Object session.

Be careful when deleting software components; do not accidentally delete business objects or libraries that are not generated from the LN Studio.

Expiring an Implementation

To expire a business object runtime, right-click the BII and select **Implementation > Expire Infor LN Implementation**. If related software projects are defined at the interface project, the generated components are expired within the used activity.

When expiring a business object runtime manually, expire the complete set of components. If you want to expire the business object ppmmm999, also expire the following libraries and additional xml file:

- ppmmmbl999sb00
- ppmmmbl999st00
- ppmmmbl999bi.xml

The latter can be expired through the **Additional Files (ttadv2570m000)** session.

Note: Do not expire a business object without expiring the corresponding libraries or additional file or vice versa. Also do not copy a business object generated from the LN Studio or one of its libraries or its additional file to another VRC. Instead, regenerate it from the LN Studio if you require the implementation to be included in another VRC.

Delivery

The runtime for a business object is delivered automatically with the application package.

Testing

General Notes

Testing the business object runtime for LN is comparable to testing the runtime for any application server. You can use the LN Studio test tool, use a test client, or use the application you are planning to integrate with.

The following section contains some notes related to aspects that are specific for LN.

Testing a Changed Implementation

Note: If you are testing a regenerated business object runtime, ensure that the previous business object runtime is not in memory anymore.

Event Publishing

Tracing

When testing event publishing it is advisable to use the tracing facility. Refer to section Troubleshooting on page 42.

Regenerating Business Object Implementations

If the business object implementation is regenerated while a publisher is running, you must unsubscribe and resubscribe. Otherwise the meta data won't match the loaded DLL objects (public/protected library).

Troubleshooting

This section gives an overview of the Implementation in LN. When troubleshooting it will be helpful to know what actually happens in LN. Additionally, the most common issues that may occur when generating are discussed.

Overview of the Implementation in LN

When generating an LN business interface implementation, the following is done in the application server:

- The business object is registered in LN. This is done to allow the application to find the implementation when at runtime a business object method is invoked. The registered business object is visible in the **Business Objects (ttadv7500m000)** session. Business objects generated from the LN Studio are marked as such.
- Two libraries are generated. The code of these libraries is derived from the implementation identifier. For example, for ppmmm999 the generated libraries will be ppmmmbl999sb00 (public interface) and ppmmmbl999st00 (protected interface). The libraries are visible in the **Program Scripts / Libraries (ttadv2530m000)** session.

Note: Do not change the generated business object or libraries in LN, because your change will be overwritten when regenerating.

The generated libraries contain a number of identifications. These can help to check which BID/BII and which generator version was used to generate the runtime.

Issues when Generating

If the generation fails, the problem is reported to the user. If a problem occurs when generating the business object runtime, it can be one of the following types:

- Problems in the connection to the application server. In this case, check whether the JCA connectivity is set up correctly. See Generating the Runtime on page 37.
- Problems in processing the template. Such problems may be caused by in incorrect model. Check whether any problems are reported by the LN Studio and whether the constraints as specified in section Modeling the Business Interface Implementation on page 16 are met.
- Problems in registering the business object in the application server, such as insufficient authorizations or an incorrect implementation identifier. Refer to section Generating the Runtime on page 37 on prerequisites for the LN application server.
- Compilation errors in the generated libraries. These may be caused by errors in the hooks (see Using Hooks for LN on page 46), problems as reported by the LN Studio, or violation of constraints as specified in section Modeling the Business Interface Implementation on page 16.

Regarding compilation errors, to have a closer look you can use the **Program Scripts / Libraries (ttadv2530m000)** session in LN to analyze the problem by compiling or viewing the libraries. You can also view the generated source in the output folder of the LN Studio project.

Tracing Event Publishing

Event publishing means actions are taken that are invisible. So when no event arrives and no error is reported, you don't know what happened. Was the event generated by the application correctly? Did the event action match the subscription? Was it skipped because of the filter or selection that was used?

To answer such questions, the tracing facility is available. When switching on tracing, a file is created showing the event publishing activity and the objects being processed. The file contents will indicate any publisher activity and any events being processed and/or sent. It will also be visible if an event message does not match the event actions, filter or selection from the subscription. The trace file includes the Show request that is used to get the component instances and attribute values for an event. In case the Show fails or gives unexpected results, you can use that request in the LN Studio Test Tool to test or debug the Show method using the same selection and filter.

To switch on tracing, add the following setting in your BW configuration:

```
-set_BOL_PUBLISHER_TRACE=/tmp/ trace_publisher
```

After the '=', specify the file where you want the trace to be written. Make sure that you have sufficient permissions to write a file in that location.

If you want to trace the detection of standard events and the actual publishing of all events, you must set this in the BW that receives the SubscribeEvent request. If you want to trace application-specific events, you must also set this in the BW where you run the application process that publishes the event. You can use two different file names, if you wish.

Note: The trace file as listed in the tlbct5500m000 session is the trace file as specified at the moment the SubscribeEvent was received. The actual tracing and trace location may be different:

- If a bshell is already running for a BDE and a SubscribeEvent request is received for the same company and destination, then the publishing for the second BDE will get the same trace settings as the first one, because it is done by the same bshell.
- If all publishers for a company are stopped and a user reactivates them through the tlbct5500m000 session, then the trace setting from that user's BW configuration is used.

Note that this allows you to change the trace settings without re-subscribing: simply start BW using the desired trace setting, then stop all publishers for a company through the tlbct5500m000 session and then reactivate them again. You need to stop all publishers for a company before reactivating them again, in order to stop the bshell! After that the user's trace settings are used.

Debugging

Just like any other code you can debug your hooks by compiling the st library in debug mode.

Note that debugging will not work if your code runs in a bshell without a user interface. To debug when running business object methods from the LN Studio test tool, set up your test connection without activation, by running the ottstpjcadaemon program. Refer to the LN Studio online help for details.

Note: If related software projects are defined for your interface project, you can set breakpoints in the generated libraries in the activity. For details, see "Creating an interface project" in the LN Studio online help.

Logging BOD Publishing

A log file for BOD Publishing can be created by adding environment variable BO_LOG_LEVEL (for example -set BO_LOG_LEVEL=debug).

Log levels are:

- Debug
- Info
- Warning
- Error

Log level Debug gives the most extensive logging and log level Error only the errors. The log is written to the file **`log.tlbct.trace`** in: `$BSE/log`

Example of a logging in the log file:

```
12-01-03[13:50:19]|DEBUG |505|ion | 55312598-15|ptlbctdisp0 | 190| Dispatch
er: invoking BOD (logical id = lid://infor.ln; message id = ID:prod.in
for.com-123456789-0:1:1673:1:1)
```

The fields mean:

Date[Time] | Severity | Company | User | Bshell Process ID - PID | Program Script | Line | Message

BOL_DISPCALLS

With the environment variable BOL_DISPCALLS it is possible to specify the number of Business Object (BOD/BDE) invocations after which the processing must be refreshed. Refresh is done by closing the subprocess when the specified number of calls has been reached. The next call will create a new subprocess. If the environment variable BOL_DISPCALLS is not set, then refresh is done after 1000 calls (for BODs) or no refresh is done at all (for BDEs), in the latter no subprocess is created and processing is done in the main process.

The reason behind BOL_DISPCALLS is that implementations of Business Objects can leak memory. After a refresh, the memory is released

BOL_AUT_PROC_WAIT

If BODs are used with automatic processing BDEs, the timeout (after which the control is given back to the caller - the BDE process(es) will continue) can be configured with environment variable

BOL_AUT_PROC_WAIT. Default timeout (envrionment variable is not set) is 60 seconds. For timeout of 2 minutes use -set BOL_AUT_PROC_WAIT=120, for immediate control given back (the old behavior) use BOL_AUT_PROC_WAIT=0.

Chapter 3: Using Hooks for LN

General Guidelines

Hook Contents

Hooks are used to implement specific behavior in a business interface implementation.

For all hooks in a business interface implementation for LN, coding is done in the language as used in LN (Enterprise Server 8) scripts and libraries.

Each hook, except the Include Hook, contains a function body. So it does not include a function header or { }.

Each hook must:

- return(0) if successful;
- use dal.set.error.message(…) and return(DALHOOKERROR) if an error occurs. (Exception: method hooks on batch methods, do not need to use dal.set.error.message() but can provide a result XML structure instead if an error occurs, see Hooks for Batch Methods on page 75).

Hooks are edited in a context-sensitive source viewer within the Business Interface Implementation editor. The source viewer offers functionality for code completion, hovering for function usage, and open declaration.

Using Libraries and Functions in Hooks

Using Libraries (DLLs)

To reuse existing business logic from LN, libraries (called ‘modules’ in LN Studio) can be used. If a library is invoked, the hook must include a declaration of the library.

For example:

```
#pragma used dll oppmmm1234
status = ppmmm1234.translate.status(internalStatus)
return(0)
```

Using Includes

Includes (called ‘functions’ in LN) can be used:

```
#include "ippmmm1234"
```

If the include is used in only one hook and if the include does not contain any functions, you can declare it in the hook where it is used. In that case, if the include contains variable declarations, those variables will become local variables for the hook.

Otherwise you can declare it in the ‘include hook’ which is linked to the BII. In that case the functions and defines from the include are available for any of the BII’s hooks.

Using Subfunctions

In most hooks, you cannot declare functions. The reason for this is that the hook content is used as the function body in the generated libraries. You can only declare functions in the ‘include hook’ which is linked to the BII. In that case the functions are available for any of the BII’s hooks. In the include hook you can also include defines which you want to reuse in multiple hooks.

Instead of using the include hook for subfunctions, you can also use library functions in any hook. For example:

```
#pragma used dll oppmmm1234
...
number2 = ppmmm1234.calculate.number(number1)
...
```

Using Global Variables

Avoid creating global variables if possible. You can declare global variables in the ‘include hook’. But when using such global variable, make sure that the logic doesn’t make uncertain assumptions about which hooks are invoked at runtime and in which sequence.

Static variables can be used. However, using static variables for optimization may be limited, because a new process may be started for handling a new request (method invocation) that arrives in LN.

Note: Never use any (global) variables that are created and used in the code as generated by the LN Implementation Generator. They are subject to change, and compatibility is never guaranteed.

Transactions

Note: Do not set retry points or execute a commit or abort in any hook. The same holds for library functions, include functions or DAL functions which are used from a hook.

Transaction handling for BODs and BDEs is done as a whole. It is uncommon to have an internal transaction. By default, any retry point, commit, or abort that is programmed within the BOD/BDE hooks is ignored by the dispatcher.

Include Hooks

An include hook can be linked to a business interface implementation. The include hook can contain any declarations (such as includes, defines or functions) that are used in one or more other hooks for the business interface implementation.

Attribute Hooks

Overview

The following table specifies the availability for attribute hooks in LN:

LN Studio	LN implementation
before get hook	see section Before/After Get/Set Hooks
on get hook	see section On Get/Set Hooks
conversion get hook	see section Conversion Hooks
after get hook	see section Before/After Get/Set Hooks
before set hook	see section Before/After Get/Set Hooks
on set hook	see section On Get/Set Hooks
conversion set hook	see section Conversion Hooks
after set hook	see section Before/After Get/Set Hooks
default value hook	see section Default Value Hooks

Note: The hooks are included in the so-called protected layer, because in the LN Studio no calculations or hooks are modeled in the BID.

Guidelines on Using or Setting Attribute Implementation Values

Introduction

In hooks, attribute (implementation) values are set or used through input and output parameters.

Available Parameters

The following parameters can be used:

- On get/set hooks always have one output parameter and zero or more input parameters. The output parameter is the attribute implementations being get/set. The input parameters are the used attributes as specified for the attribute implementations being get/set.
- Before get/set hooks have one output parameter io.cancel, and zero or more input parameters for the used attributes.
- After get/set hooks only have input parameters, being the used attributes.
- Conversion get/set hooks have one input parameter and one output parameter. For a conversion get hook, the input parameter is the value from the attribute implementation, and the output parameter is the value for the corresponding attribute. For a conversion set hook, the input parameter is the value for the attribute, and the output parameter is the value for the corresponding attribute implementation.

The attribute implementation names are used in the parameter names.

The parameters are named as follows:

Hook	Available Parameters	Variable Names
before/on/after get/set	Input parameters for used attributes	If the attribute implementation names of the used attributes are orderNumber and businessPartner: i.orderNumber, i.businessPartner
before get/set	Input/output parameter for cancel	io.cancel
on get/set	Output parameter for attribute being get/set	If the name of the attribute implementation being retrieved/set is orderQuantity: o.orderQuantity
after get/set	Input parameter for attribute being get/set	If the name of the attribute implementation being retrieved/set is orderQuantity: i.orderQuantity

Hook	Available Parameters	Variable Names
default value	Input/output parameter for attribute implementation, and input parameters for used attributes (if any)	if the name of the attribute implementation is orderType: io.orderType, if the attribute implementation names of the used attributes are orderNumber and businessPartner: i.orderNumber, i.businessPartner
conversion get	Input parameter for attribute implementation; output parameter for corresponding attribute	if the name of the attribute implementation is orderType: i.orderType, o.orderType
conversion set	Input parameter for attribute; output parameter for corresponding attribute implementation	if the name of the attribute implementation is orderType: i.orderType, o.orderType
filter	Output parameter for filter	o.hookFilter

The data type of the parameters is the same as the data type of the attribute implementations involved.

In case attribute grouping is used, the attribute implementation names by default contain the path, using underscores. For example, if the following structure exists:

Component ‘Order’ -> group ‘Customer’ -> group ‘Address’ -> attribute ‘city’

then the attribute implementation name will be:

‘Customer_Address_city’

Global Variables

Note: Never use or set global variables directly from a hook. This will lead to errors. The LN Implementation Generator will make sure that the used values for the parameters are available, but will not make sure that all other attribute values are set correctly in all cases.

Additional Information in Attribute Hooks

The calculation of an attribute implementation may differs depending on whether attributes are specified in the request or not. However, an empty value can indicate either the (optional) attribute value was unavailable in the request, or the attribute value was available but happened to be empty. For that reason, the developer who is coding the hook must be able to determine which input attributes are set.

In before/on/after get/set hooks a parameter is available to indicate whether the value for an attribute implementation is set. Note that an on set hook is invoked if at least one of the used attributes is set. So if multiple used attributes exist, some of those may not be set. However, if none of the used attributes is set then the setter is not invoked. By the way, if an attribute has no used attributes at all, the on set hook is always invoked.

For the attribute implementation that is output for a hook, a parameter is available to indicate whether the value is set. By default the value will be true; so it only needs to be set if false.

An example of using an ‘isSet’ parameter in a before set hook:

```
if not i.orderStatus.isSet then
   io.cancel = true
endif
return(0) | OK
```

An example of using ‘isSet’ parameters in an on set or on get hook:

```
if i.value1.isSet then
   if i.value2.isSet then
      o.calculatedValue = i.value1 * i.value2
   else
      o.calculatedValue = i.value1 * i.value3
   endif
else
   io.calculatedValue.isSet = false
endif
return(0) | OK
```

On Get/Set Hooks

Guidelines

Note: In on get/set hooks for an attribute implementation, only use attributes that are modeled as ‘used attributes’ for that attribute implementation!

Note that you cannot have both an on set hook and an on get hook on the same attribute implementation.

An attribute implementation value can be filled (or changed) only in the following cases:

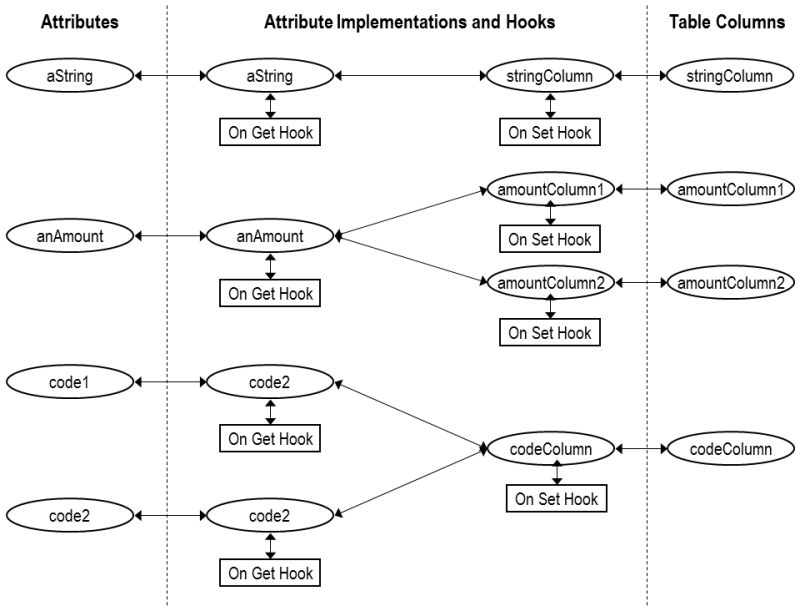
- In case of an on set hook, if it is the attribute implementation that contains the on set hook.
- In case of an on get hook, if it is the attribute implementation that contains the on get hook.

An attribute implementation value can be used only in the following cases:

- In case of an on set hook, if it is one of the attribute implementations listed in the used attributes of the attribute implementation containing the on set hook.
- In case of an on get hook, if it is one of the attribute implementations listed in the used attributes of the attribute implementation containing the on get hook.

Examples

The following figure gives an example that contains a one-to-one, a one-to-many and a many-to-one relationship for attributes and table columns.



Note: In the LN Business Object Repository (BOR), the predecessor of the on set hook was not linked to the attribute being set, but to the attribute that also had the on get hook. This is different in the LN Studio.

The mapping between the attributes and table columns can be as follows:

Set	Get
stringColumn = aString(4)	aString = “ERP_” & stringColumn
calculateInternalAmountColumn1(anAmount, amountColumn1); calculateInternalAmountColumn2(anAmount, amountColumn2)	calculateExternalAmount(amountColumn1, amountColumn2, anAmount)
determineInternalCode(code1, code2, codeColumn)	determineExternalCode1(codeColumn, code1); determineExternalCode2(codeColumn, code2)

Since we have got two separate on set hooks for amountColumn1 and amountColumn2, reuse of existing library functions has a drawback. For example, assume we have got an existing function calculateInternalAmounts(anAmount, amountColumn1, amountColumn2), we can use it from both the amountColumn1 on set hook and the amountColumn2 on set hook. But note that it will be executed twice if anAmount is set and consequently amountColumn1 and amountColumn2 are set.

For the example in the previous figure, the hooks are as follows:

On Get hook for aString:

```
| add prefix "ERP_"
o.aString = "ERP_" & i.stringColumn
return(0) | OK
```

Note that it is not needed to check whether stringColumn is set or not. If the model is OK, stringColumn is listed as a used attribute for the calculation, so the standard mechanism will take care that the required attribute is set. This also is the case for the following hooks:

On Set hook for stringColumn:

```
| remove prefix "…_", for example "ERP_"; prefix will always be 3 charac
ters plus underscore
if len(strip$(i.aString) < 4 then
 dal.set.error.message("ppmmms1234.01", i.aString)
 |* Component.aString %s does not have the required prefix (such as "ERP_")

 return(DALHOOKERROR)
else
 o.stringColumn = i.aString(4)
 return(0) | OK
endif
```

On Get hook for anAmount:

```
#pragma used dll oppmmmamounts
| library ppmmmamount contains:
| function long ppmmmamount.calculateExternalAmount(
|    double  i.amountColumn1,
|    double  i.amountColumn2,
|  ref  double  o.anAmount)

long retl  | return value to be checked

retl = ppmmmamount.calculateExternalAmount(i.amountColumn1, i.amountCol
umn2,
           o.anAmount)
if retl <> 0 then
 dal.set.error.message("ppmmms1234.02", i.amountColumn1, i.amountColumn2)

 |* Cannot calculate Component.anAmount for amountColumn1 = %f and
amountColumn2 = %f
 return(DALHOOKERROR)
else
 return(0) | OK
endif
```

On Set hook for amountColumn1:

```
#pragma used dll oppmmmamounts
| library ppmmmamount contains:
| function long ppmmmamount.calculateInternalAmountamountColumn1(
```

```
|    double  i.anAmount,
|  ref  double  o.amountColumn1)

long retl  | return value to be checked

retl = ppmmmamount.calculateInternalAmountamountColumn1(i.anAmount,
           o.amountColumn1)
if retl <> 0 then
 dal.set.error.message("ppmmms1234.03", i.anAmount)
 |* Cannot calculate Component.amountColumn1 for anAmount = %f
 return(DALHOOKERROR)
else
 return(0) | OK
endif
```

On Set hook for amountColumn2:

```
#pragma used dll oppmmmamounts
| library ppmmmamount contains:
| function long ppmmmamount.calculateInternalAmountamountColumn2(
|    double  i.anAmount,
|  ref  double  o.amountColumn2)

long retl  | return value to be checked

retl = ppmmmamount.calculateInternalAmountamountColumn2(i.anAmount,
           o.amountColumn2)
if retl <> 0 then
 dal.set.error.message("ppmmms1234.04", i.anAmount)
 |* Cannot calculate Component.amountColumn2 for anAmount = %f
 return(DALHOOKERROR)
else
 return(0) | OK
endif
```

On Get hook for code1:

```
#pragma used dll oppmmmcoding

long retl  | return value to be checked

retl = ppmmmcoding.determineExternalCode1(i.codeColumn, o.code1)
if retl <> 0 then
 dal.set.error.message("ppmmms1234.05", i.codeColumn)
 |* Cannot determine Component.code1 for codeColumn = %s
 return(DALHOOKERROR)
else
 return(0) | OK
endif
```

On Get hook for code2:

```
#pragma used dll oppmmmcoding
```

```
long retl  | return value to be checked

retl = ppmmmcoding.determineExternalCode2(i.codeColumn, o.code2)
if retl <> 0 then
 dal.set.error.message("ppmmms1234.06", i.codeColumn)
 |* Cannot determine Component.code2 for codeColumn = %s
 return(DALHOOKERROR)
else
 return(0) | OK
endif
```

On Set hook for codeColumn:

```
#pragma used dll oppmmmcoding

long retl  | return value to be checked

retl = ppmmmcoding.determineInternalCode(i.code1, i.code2, o.codeColumn)
if retl <> 0 then
 dal.set.error.message("ppmmms1234.07", i.code1, i.code2)
 |* Cannot determine Component.codeColumn for code1 = %s and 4 = %s
 return(DALHOOKERROR)
else
 return(0) | OK
endif
```

Conversion Hooks

Guidelines

Conversion hooks can only be used to convert a value from the data type used in an attribute to the data type used in the corresponding attribute implementation, and vice versa.

Only one input parameter and one output parameter are used. Other attribute implementations or table columns must not be used in a conversion hook. The parameter representing the attribute is always a string. The parameter representing the corresponding attribute implementation is declared on its domain.

The actual conversion is done in the public layer. So when using the protected interface within LN, only the domains as specified for the attribute implementations are used. For example, if you define a conversion hook for an enumerate, the attribute will have a string data type, but the attribute implementation will use the enumerate domain as its data type.

Standard data type conversions (see Standard Conversions on page 56) need not be implemented in a hook.

Example

A conversion get hook for an itemCode attribute can contain the following:

```
o.itemCodeAttribute = toupper$(i.itemCodeImplementation)
return(0) | OK
```

And the corresponding conversion set hook can contain:

```
o.itemCodeImplementation = tolower$(i.itemCodeAttribute)
return(0) | OK
```

Standard Conversions

Overview

In some cases, data types used internally in LN differ from standard formats as used in the BIDs. In a number of cases an automatic data type conversion is done, to avoid programming efforts and risks of errors.

Standard conversions are available for:

- Boolean
- UTC date/time and date
- Enumerate
- Text
- Number to string and vice versa
- Doubles
- XML.

Each of the listed item is explained later. No other standard conversions are available.

Note: Standard conversions can be overruled by implementing a conversion hook. So when defining a conversion hook, the standard data type conversion is not executed.

Just like calculations, conversion hooks impact performance, because a filter on a calculated value cannot be delegated to the DBMS. Instead too much data is read from the database, and ‘postfiltering’ is done after calculating the public attribute value, which is an expensive process. So if possible, avoid conversion hooks, especially on enumerate values.

Boolean Conversions

A standard Boolean conversion is done if:

- The attribute is a Boolean, but the corresponding attribute implementation has primitive data type ‘long’.
- The attribute is a boolean, but the corresponding attribute implementation is a ‘boolean enumerate’, i.e. a string data type having two enumeration facets: “yes”/“no” or “Yes”/“No” or “true”/“false” or “True”/“False”.

UTC and Date Conversions

Automatic conversion for UTC date/time values is done if the attribute implementation that is linked to the attribute has primitive data type ‘time’ or ‘dateTime’. The corresponding attribute implementation is then assumed to have a UTC date/time domain as its native data type.

From outside to inside, a string according to the ISO 8601 format is converted to a UTC date/time according to the internal LN representation. From inside to outside, the internal LN representation is converted to an ISO 8601 string.

In this case, the used ISO 8601 format is any of the following:

- yyyy-mm-ddThh:mm:ssZ
- yyyy-mm-ddThh:mm:ss+hh:mm
- yyyy-mm-ddThh:mm:ss-hh:mm

Automatic conversion for date values is done if the attribute implementation that is linked to the attribute has primitive data type ‘date’. The corresponding attribute implementation is then assumed to have a date domain as its native data type.

From outside to inside, a string according to the yyyy-mm-dd format is converted to a date in the internal LN representation. From inside to outside, the internal LN representation is converted to a date string having the yyyy-mm-dd format.

Enumerate Conversions

Automatic conversion for enumerate values is done if the attribute has primitive data type ‘string’ and one or more enumeration facets. The corresponding attribute implementation is then assumed to have a matching enumerate domain as its native data type.

Note: The standard enumerate conversion (based on the enumerate constants as defined in the LN domain) uses lower case. So if possible, use lower-case enumerate values in your BID, matching the definition of the enumerate domain.

Text Conversions

For texts, a default conversion is available from the text number as used in LN to the text contents (string) as used in the public attributes. This decision is made based on the **DB Type** property of the attribute implementation. If it is “db.text” then a default text conversion is done and if it has another value the conversion will not be done.

However, since the DB Type is options, the generator in the LN Studio may not have all native domain information from LN available in the data types. Consequently, an assumption is made regarding which data type is actually a text.

If the DB Type is empty, the following rule is used:

If the data type's native data type (which contains the domain) is ?????.text or ??text or ?????.txtn or ??txtn or ?????.txta or ??txta, where '?' can be any character, then the data type is assumed to indicate a text.

This means that domains that do not match this rule will be handled incorrectly and will result in failure when generating the LN implementation or at runtime. For example, if a domain is used that matches one of the patterns (for example pptext) but is not a text domain, or if a text domain is used that does not match one of the patterns (for example ppmytext).

If such a problem occurs, the solution is to fill in the DB Type or use another domain as the native data type.

Number/String Conversions

Automatic conversion between numeric values and string values is done if:

- the attribute is a string, but the corresponding attribute implementation is numeric (long, double or float)
- the attribute is numeric (long, double or float), but the corresponding attribute implementation is a string.

In this case,

- String means: primitive data type in {"string", "anyURI", "duration", "Name"}
- Long means: primitive data type in {"byte", "int", "integer", "negativeInteger", "nonNegativeInteger", "nonPositiveInteger", "normalizedString", "positiveInteger", "short", "unsignedByte", "unsignedInt", "unsignedLong", "unsignedShort", "long"}
- Float means: primitive data type in {"float"}
- Double means: primitive data type in {"decimal", "double"}

Double Values

In case both the attribute and the attribute implementation have primitive data type ‘double’ no conversion is needed. However, double values are discussed here because they can have multiple formats.

The following specifies which values are accepted as input:

- <validDouble> ::= [whiteSpace]<significand>[<exponent>][whiteSpace]
- <significand> ::= [<sign>]<unsignedSignificand>
- <unsignedSignificand> ::= <number> | “.”<number> | <number>“.”<number>
- <exponent> ::= “e”|“E” [<sign>]<number>
- <sign> ::= “+”|“-”
- <number> ::= <digit> | <digit><number>

- <digit> ::= “0”|“1”|“2”|“3”|“4”|“5”|“6”|“7”|“8”|“9”
- <whiteSpace> ::= “ ” | “ ”<whiteSpace>

As specified earlier, leading and trailing spaces are accepted. Spaces inside significand or exponent or between significand and exponent are not allowed.

Examples of valid doubles:

"12", "12.3", "0.3", ".3", "+12", "+12.3", "+.3", "-12", "-12.3", "-.3", "012", "012.0".

Each of these can be followed by an exponent. So other valid doubles are for example:

"12.3e1", ".3E2", "+12.3e+12", ".3e-12", "-12E+12", "1E-12".

Examples of invalid doubles:

"12,3", "1,000", "1.000,25", "100e", "12.", ".", "e3", "12 e3" (contains a space), "12e0.5"

Note: The "INF" value is valid according to W3C, but is regarded as invalid; it is not supported. See also http://www.w3.org.

XML Conversions

Automatic conversion for XML is done if the attribute and the corresponding attribute implementation have primitive data type ‘anyType’. The attribute implementation must then have a ‘long’ domain as its native data type, which refers to an in-memory XML structure.

For an ‘anyType’, a hook must always be implemented. Mapping the attribute implementation directly to a table column will fail. In that case, when setting or changing the value, the column value will be the number that used to be the reference to the xml structure, but the actual xml structure will be lost.

Default Value Hooks

Guidelines

A default value hook is comparable to an on set hook.

In a default value hook you must set the attribute to which the default value hook is linked. This variable is not declared in the hook code. You can only set one attribute implementation in a default value hook.

The attribute implementation’s used attributes can be used to determine the default value.

A default value hook can be used if the default value is not a constant value. For example, if it is based on used attributes or on a parameter value from the database. In other cases, a simple default value can be linked to the attribute implementation.

Note that an attribute implementation can have both a default value and a default value hook. In that case the default value hook has precedence over the default value. The default value is only used if the default value hook doesn’t overwrite its value.

Example

```
#pragma used dll oppmmmparam
long  retl  | return value to be checked
domain ppmmm.opar   order.parameter

retl = ppmmmparam.get.order.parameter(i.customer, order.parameter)
if retl <> 0 then
 dal.set.error.message("ppmmms1234.11", i.customer)
 |* Order parameter is not set for customer %s
 return(DALHOOKERROR)
else
 if order.parameter = ppmmm.opar.operational then
  io.orderType = ppmmm.otyp.production
 else
  io.orderType = ppmmm.otyp.virtual
 endif
 return(0) | OK
endif
```

In case a default value is linked to the attribute, that default will be set before executing the default value hook.

For example, if the default value for attribute ‘myAttribute’ is 1, the default value hook can contain the following:

```
if some.condition() then
 io.myAttribute = 2
| else
| default value linked to attribute implementation (for example 1) is used
endif
return(0) | OK
```

Before/After Get/Set Hooks

Guidelines

In before and after get/set hooks, the same attributes can be used that are input for on get/set hooks. Refer to section On Get/Set Hooks on page 51 on how to use them.

Before and after get/set hooks do not have any output, except for an io.cancel in the before get/set hook. It is not allowed to change business object attributes are the corresponding persistent data in the database in a before/after get/set hook.

Only use io.cancel if it is not a problem that the attribute is skipped, and the client application does not need to be informed about this. In case of an exception, do not use io.cancel, but set an error message and return DALHOOKERROR instead.

Examples

Two hypothetical examples of before/after set hooks for a ‘price’ attribute.

Before set hook:

```
| before set hook for attribute price; only set this attribute if status
 and type allow this

| ‘used attributes’ i.order and i.price are unused in this before set hook

long retl  | return value to be checked

if i.type = ppmmm.type.external and i.status = ppmmm.stat.confirmed then
 | do not set price attribute
 io.cancel = true
endif
return(0) | OK
```

After set hook:

```
| after set hook for attribute price; only set this attribute if status
and type allow this

| ‘used attributes’ i.status and i.type are unused in this after set hook

#pragma used dll oppmmmsox

if i.price > 999999.0 then
 ppmmmsox.report.high.price(i.order)
endif
return(0) | OK
```

Hooks for Repeatable Attributes

When an attribute implementation having dimensions is used in a hook, the interface for the attribute hooks will be different.

For set/get hooks, the interface is as follows. For example, let’s assume an on set hook is defined for an attribute implementation ‘note’ having dimension 5. Three used attributes are specified, the first one ‘optionalNote’ has no dimension and the other ones (‘extraNotes’ and ‘numbers’) have dimension 2. The last one (‘numbers’) is a long array, while the other ones are strings.

In that case the hook content for the on set hook can for example be:

```
 if i.optionalNote.iSet then
  o.note(1,1) = i.optionalNote
 else
  io.note.isSet(1) = false
 endif
 o.note(1,2) = i.extraNotes(1,1)
```

```
o.note(1,3) = i.extraNotes(1,2)
o.note(1,4) = str$(i.numbers(1))
o.note(1,5) = str$(i.numbers(2))
return(0) | OK
```

Note that strings have an extra dimension.

Hooks for ‘anyType’ Attribute Implementations

Note:

- An anyType attribute must be linked to an anyType attribute implementation, having a native datatype (domain) which is a 'long'.
- An anyType can have dimensions.

Hook Output for ‘anyType’ Attribute Implementations

In an on get hook or on set hook for an 'anyType', either the 'isSet' must be set to false and the output must be 0, or the isSet must be true and the output must contain a valid XML where the top-level node uses the attribute name.

For example, in the BID an attribute group 'Group' contains an anyType attribute 'Attribute'. Let’s assume in the BII an on get hook exists for the corresponding Group_Attribute attribute implementation.

In that case the on get hook will have the following parameters:

- ref boolean io.Group_Attribute.isSet (default is true)
- ref long o.Group_Attribute

The following table shows examples of valid and invalid combinations for on get hooks for the Group_Attribute attribute implementation.

Valid	Invalid
io.Group_Attribute.isSet = false o.Group_Attribute = 0	o.Group_Attribute = 0
o.Group_Attribute = xmlNewNode("Attribute")	io.Group_Attribute.isSet = false o.Group_Attribute = xmlNewNode("Attribute")
o.Group_Attribute = xmlReadFromString("<Attribute><aGroup>” & “<value1>x</value1></aGroup>” & “<value2>y</value2></Attribute>", error.string)	o.Group_Attribute = xmlNewNode("aNode")

(Of course the hook must also include error handling where applicable and a return statement, as usual.)

If the on get hook returns the following XML in o.Group_Attribute:

```
<Attribute>
  <aGroup>
    <value1>x</value1>
  </aGroup>
  <value2>y</value2>
</Attribute>
```

then the 'Group' element in the response or event will contain that XML. So the top-level node from the on get hook will be the node representing the ‘anyType’ attribute.

Hook Input for ‘anyType’ Used Attributes

If an anyType attribute implementation is a used attribute for another attribute implementation then the following happens.

For example, Group_Attribute is a used attribute for another attribute implementation InternalAttribute having an on set hook.

If the request contains:

```
<Group>
  <Attribute>
    <aGroup>
      <value1>x</value1>
    </aGroup>
    <value2>y</value2>
  <Attribute>
<Group>
```

then in the on set hook for InternalAttribute, the i.Group_Attribute parameter will contain a reference to the Attribute node:

```
<Attribute>
  <aGroup>
    <value1>x</value1>
  </aGroup>
  <value2>y</value2>
<Attribute>
```

and i.Group_Attribute.isSet will be true

If the request does not contain the Attribute node then in the on set hook:

- i.Group_Attribute = 0 and
- i.Group_Attribute.isSet = false

Filter Hooks

Guidelines

Note: The filter hook content is not yet compliant to the BI standard.

A filter hook usually consists of two parts:

- The assignment of the output parameter o.hookFilter (the actual filter): a string expression is assigned, which is a query extension. In this expression, any columns from the component's root table can be used.
- The 'return' at the end. Usually a filter hook will not return an error value.

Examples

Assuming a component has root table ppabc001, the filter hook can contain:

```
o.hookFilter = "ppabc001.col1 <> 0 and ppabc001.col2 <> ppabc.stat.can
celed"
return(0) | OK
```

A real-life example:

```
| do not include canceled orders
o.hookFilter = "tdsls400.clyn <> tcyesno.yes"
return(0) | OK
```

Method Hooks

Overview

The following table shows the available method hooks:

LN Studio	LN implementation
before execute hook	see section Before Execute Method Hook on page 71
on execute hook	see section On Execute Method Hook on page 73
after execute hook	see section After Execute Method Hook on page 74

Specifics on using hooks for 'batch' implementations are discussed in section Hooks for Batch Methods on page 75.

The following table indicates which hooks are supported for LN:

Method	Hooks		
	Before execute	On execute	After execute
List	Yes	Yes	Yes
Show	Yes	Yes	Yes
Create	Yes	Yes	Yes
Change	Yes	Yes	Yes
Delete	Yes	Yes	Yes
SubscribeEvent	Yes	Yes	Yes
UnsubscribeEvent	Yes	Yes	Yes
PublishEvent	Yes	Yes	Yes
OnEvent	Yes	Yes	Yes
SubscribeList	No	No	No
SupportsProcessingScope	No	No	No
SupportsReferentialIntegrity	Yes	Yes	Yes
CreateRef	Yes	Yes	Yes
DeleteRef	Yes	Yes	Yes
Any specific method (including methods for receiving BODs, see Implementing OAGIS BODs for LN on page 132)	Yes	Yes (Mandatory)	Yes

For most standard methods, if you create an on execute hook it is possible to fall back to the default behavior. This is done through io.default (see section On Execute Method Hook on page 73).

The following table shows for what methods io.default can be used.

Method	io.default if top-down/bottom-up	io.default in batch implementation	Notes
List	no	no	(1)
Show	no	no	(1)
Create	yes	no	(2)
Change	yes	no	(2)
Delete	yes	no	(2)
SubscribeEvent	n/a	yes	

Method	io.default if top-down/bottom-up	io.default in batch implementation	Notes
UnsubscribeEvent	n/a	yes	
PublishEvent	n/a	yes	
OnEvent	n/a	yes	
SubscribeList	n/a	n/a	(3)
SupportsProcessingScope	n/a	n/a	(3)
SupportsReferentialIntegrity	n/a	yes	
CreateRef	yes	no	
DeleteRef	yes	no	

1 io.default is unsupported for List and Show.

2 For methods that are by default handled top-down or bottom-up, no standard batch implementation is available.

3 Hooks are unused for this method.

General Guidelines

The guidelines mentioned earlier in section General Guidelines on page 46 are also valid for method hooks, except for transaction management.

Database transactions (commit/abort) are needed in hooks for methods having a batch implementation, if the processingScope is ‘business_entity’ or ‘business_entity_component’.

Specific method (including methods for receiving BODs, see Implementing OAGIS BODs for LN on page 132) must have an on execute hook.

For method hooks in List and Show methods, refer to section List and Show Methods on page 89.

Guidelines on Using or Setting Attribute Implementation Values

Important Notes

- A ‘batch’ method does not have the attribute values available as parameters. For details refer to section Hooks for Batch Methods on page 75.
- For LN, controlling attributes cannot be used as method arguments.

Using Method Arguments for Top-Down and Bottom-Up Methods

Attribute implementation values being input, output or both, are listed as method arguments for a method. Additionally, related attributes (which are derived from the input arguments or used to calculate the output arguments are also available in hooks. This is specified in the next subsection.

The naming of attribute implementations as used in method hooks is comparable to the naming as used in attribute hooks (see section Guidelines on Using or Setting Attribute Implementation Values on page 49). In other words, the attribute implementation names are used, and a prefix (‘i.’, ‘o.’ or ‘io.’) indicates whether the value is input, output or both. However, for attribute hooks it is predefined which attributes are inputs, output or input/output, but for method hooks this depends on the settings of the method argument implementations.

Let’s assume a method exists having the following attribute implementations listed as method argument implementations:

- orderNumber, input: i.orderNumber
- businessPartner, input: i.businessPartner
- deliveryDate, input/output: io.deliveryDate
- orderStatus, output: o.orderStatus
- deliveryAddress, output: o.deliveryAddress

The following table shows what parameters for these attribute implementations can be used in before/on/after execute hooks for this method.

Hook	Arguments
before execute	Input and input/output arguments can be used, read-only: i.orderNumber, i.businessPartner, i.deliveryDate Output arguments cannot be used (technically, parameters i.orderStatus and i.deliveryAddress may be available, but they will not be filled)
on execute	Input arguments can be used, read-only: i.orderNumber, i.businessPartner Input/output arguments can be used and set in the hook: io.deliveryDate Output arguments are set in the hook: o.orderStatus, o.deliveryAddress
after execute	Input, input/output and output arguments can be used, all are read-only: i.order-Number, i.deliveryDate, i.orderStatus, i.businessPartner, i.deliveryAddress

In case attribute grouping is used, the attribute implementation names by default contain the path, using underscores. For example, if the following structure exists:

Component ‘Order’ -> group ‘Customer’ -> group ‘Address’ -> attribute ‘city’

then the attribute implementation name will be:

‘Customer_Address_city’.

Note: It is not allowed to change any attribute implementation values in a before or after execute hook.

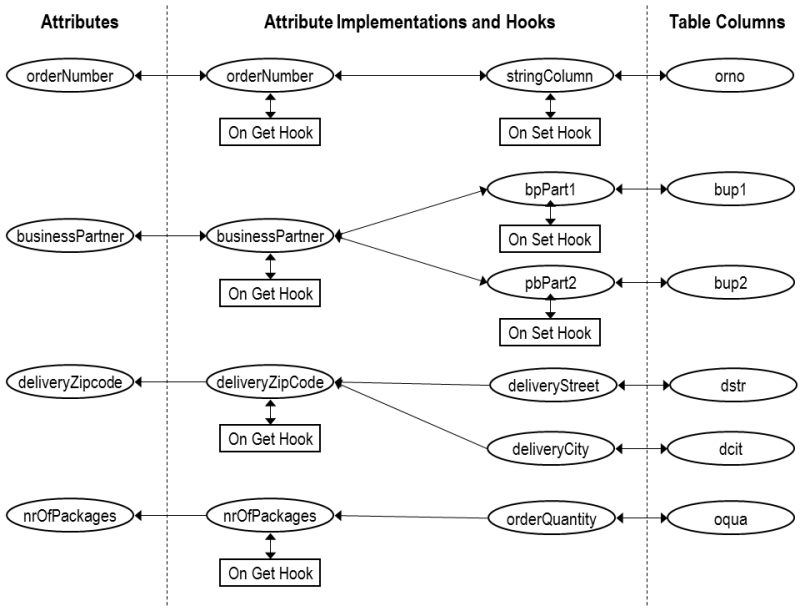
Calculated Attributes and Method Hooks

In case attribute implementations are derived from the method implementation input arguments or attribute implementations are used to calculate the method implementation output arguments then the related attribute implementations are also available as input/output.

Note: It is not needed to specify the related protected attribute implementations as method argument implementations explicitly. The only case in which protected attribute implementations must be used as method argument implementations is for the identifying attributes of subcomponents. For example, an orderNumber attribute implementation in an orderLine component.

Table columns cannot be used directly, unless the table is not mapped to the component, but read or updated within the hook.

Let's assume a method has orderNumber and businessPartner as input, and deliveryAddress and numberOfPackages as output. The attribute implementations are defined, as shown in the following figure:



In that case the following parameters for attribute implementations are available in the method hooks:

Hook	Parameters
before execute	i.orderNumber, i.stringColumn, i.businessPartner, i.bpPart1, i.bpPart2 i.orderNumber.isSet, i.stringColumn.isSet, i.businessPartner.isSet, i.bpPart1.isSet, i.bpPart2.isSet
on execute	i.orderNumber, i.stringColumn, i.businessPartner, i.bpPart1, i.bpPart2, o.deliveryZipCode, o.deliveryStreet, o.deliveryCity, o.nrOrPackages, o.orderQuantity i.orderNumber.isSet, i.stringColumn.isSet, i.businessPartner.isSet, i.bpPart1.isSet, i.bpPart2.isSet o.deliveryZipCode.isSet, o.deliveryStreet.isSet, o.deliveryCity.isSet, o.nrOrPackages.isSet, o.orderQuantity.isSet

Hook	Parameters
after execute	i.orderNumber, i.stringColumn, i.businessPartner, i.bpPart1, i.bpPart2, i.deliveryZipCode, i.deliveryStreet, i.deliveryCity, i.nrOrPackages, i.orderQuantity i.orderNumber.isSet, i.stringColumn.isSet, i.businessPartner.isSet, i.bpPart1.isSet, i.bpPart2.isSet i.deliveryZipCode.isSet, i.deliveryStreet.isSet, i.deliveryCity.isSet, i.nrOrPackages.isSet, i.orderQuantity.isSet

Note: When values are set, avoid redundancy. For example, when setting o.orderQuantity, do not set o.nrOfPackages and vice versa.

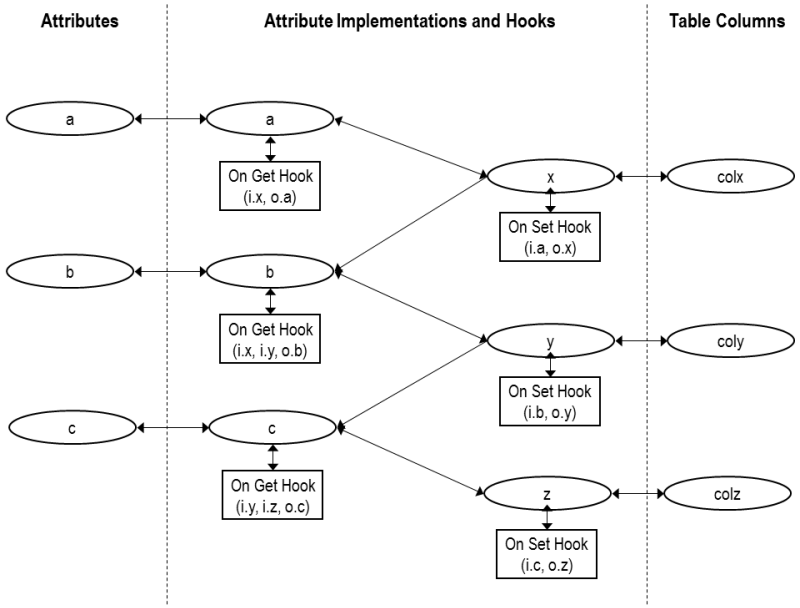
When setting an output value (or an input/output value that was not yet set), also set the corresponding o.<attribute>.isSet (or io.attribute.isSet) to true. If desirable, you can use a define for this, such as:

```
#define setValue(attr_impl, value) attr_impl = value
^          attr_impl##.isSet = true
```

You can use for example:

- setValue(o.deliveryStreet, some.street)
- setValue(o.deliveryCity, some.city)

In some cases, the same protected attribute implementations are used for multiple attributes. And each of these attributes can be input, output or input/output. An example is shown in the following figure:



In case attribute a is input, attribute b is input/output and attribute c is output, the following parameters will be available in the method hooks:

Hook	Parameters
before execute	i.a, i.b, i.x, i.y i.a.isSet, i.b.isSet, i.x.isSet, i.y.isSet
on execute	i.a, io.b, o.c, io.x, io.y, o.z i.a.isSet, i.b.isSet, o.c.isSet, i.x.isSet, io.y.isSet, o.z.isSet
after execute	i.a, i.b, i.c, i.x, i.y, i.z i.a.isSet, i.b.isSet, i.c.isSet, i.x.isSet, i.y.isSet, i.z.isSet

There won't be any duplicate attribute implementations in the parameters. Two or three of the following variants

will not occur at the same time. Instead, a single io.something will be used. The LN Implementation Generator will simply merge the method arguments and their derived attributes. In the example earlier, if the developer explicitly (and incorrectly) adds x as an output argument, this will cause 'i.x' in the on execute hook to be changed to io.x.

Note: Again, do not forget to set the isSet parameters as applicable.

For example:

ppmmmdll1234.calculate(i.a, i.b, io.c, o.d)

- io.c.isSet = true
- o.d.isSet = true

Using Standard Control Data

Standard control data is not defined explicitly as input or output for a method.

Controlling attributes (either standard or specific controlling attributes) are not passed on as parameters for hooks. Instead, control data can be used through the getControlAttribute() helper method. Refer to section Assisting Functions on page 78.

It is not possible to update standard control data from the input in a hook.

Note: For LN, controlling attributes cannot be used as method arguments. Consequently, controlling attributes also cannot be output for a method implementation. Controlling attributes cannot be set.

Exception is when using specific methods having processing order 'batch', because then you must build up the whole response in the on execute hook.

See Hooks for Batch Methods on page 75.

In that case note however that the control area is used for the request/response as a whole, so no controlling attributes can be determined per object instance.

Additional Information in Method Hooks

In a number of cases the business logic differs depending on whether attributes are specified in the request or not. However, an empty value can indicate either the (optional) attribute value was unavailable in the request, or the attribute value was available but happened to be empty. For that reason, the developer who is coding the hook must be able to determine which input attributes are set.

In before, on and after execute method hooks, for each attribute implementation that is input for a method hook, a parameter is available to indicate whether the value is set. The parameter is true if the value is set (either from the request that was input for the method or through a default value or default value hook).

In on execute hooks, for each attribute implementation that is output for a hook, a parameter is available to indicate whether the value is set. By default the value is false, so it must be set for each attribute implementation attribute that is set in the hook.

An example of using an ‘isSet’ parameter in a before execute hook:

```
if not i.orderStatus.isSet then
 io.cancel = true
endif
return(0) | OK
```

An example of using ‘isSet’ parameters in an on execute hook:

```
#pragma used dll oppmmmdll
return( ppmmmdll.my.function(i.orderNumber, i.orderNumber.isSet,
        i.businessPartner, i.businessPartner.isSet,
        io.orderStatus, io.orderStatus.isSet,
        o.deliveryDate, o.deliveryDate.isSet) )
```

Note: In an on execute method hook, when setting o.deliveryDate or io.orderStatus, also set o.deliveryDate.isSet or io.orderStatus.isSet to true!

The ‘isSet’ parameters are unavailable for batch implementations.

Before Execute Method Hook

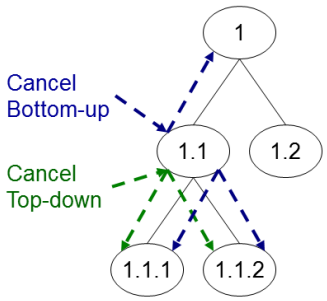
Guidelines

If the method has processing order ‘batch’ then the on execute hook is linked to the top-level component (the business object level). In that case the whole request must be used. In other cases, only the attribute values from the current component and identifying attributes from the parent components (if any) can be used.

In a before execute hook it is not allowed to change the object instance being processed (either its attribute values or its persistent data in the database).

Output parameter io.cancel can be set. This is a boolean that indicates whether the method execution must be cancelled. If it is not set, the default value is false.

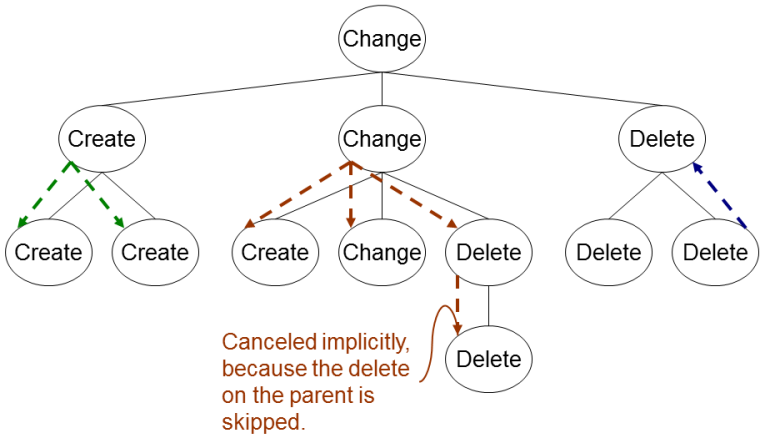
If a business object consists of multiple components, the cancel is handled as shown in the following picture.



In case of a bottom-up method (for example Delete), if a cancel occurs on a component instance, the method is not executed for that component and its parents. A cancel on 1.1 results in skipping the method for 1.1 and 1, but also for 1.1.1 and 1.1.2. So 1.2 is processed. In case of a top-down method (for example Create), if a cancel occurs on a component instance, the method is not executed for that component and its children. A cancel on 1.1 results in skipping the method for 1.1, 1.1.1 and 1.1.2.

These rules apply to standard and specific methods.

In case of a Change method, the situation is not that straightforward, because a change combines the top-down and the bottom-up approach. A change may include create or delete actions on subcomponents. A cancel on a delete for a child, does not impact the change on the header! Three cancel situations are shown in the following example.



Example

```
if i.orderDirection = ppmmm.stat.inbound then
 io.cancel = true
 | skip inbound orders in this method
endif
return(0) | OK
```

On Execute Method Hook

Guidelines

If the method has processing order ‘batch’ then the on execute hook is linked to the top-level component (the business object level). In that case the whole request must be used. In other cases, only attribute values from the current component and identifying attributes from the parent components (if any) can be used.

From an on execute hook for a standard method, it is possible to fall back to the default behavior (except for List and Show). This is done by setting output parameter io.default to true and returning the OK value (0).

When generating an LN implementation, protected methods and protected BIIs will not only get a protected interface in the generated ‘st’ library, but also will also have an XML-based interface in the generated ‘sb’ library. This way, you can forward a request XML from a public method’s on execute hook to a protected method of the same or another business object. See section Examples on page 77 about sending and receiving a BOD for an example.

Example

On execute hook for Create (processing order ‘top-down’):

```
#pragma used dll oppmmmorderline  | my own order line implementation

return( ppmmmorder.create.order.line(i.orderNumber, i.item, i.quantity,
i.price, o.lineNumber) )
```

After Execute Method Hook

Guidelines

The after execute hook is comparable to the before execute hook, except that:

- no io.cancel is available;
- the output (and input/output) attribute implementations now have their values set, because the method is executed, so these values can be used in the after execute hook.

See earlier section, Before Execute Method Hook on page 71 for guidelines.

Example

```
| after execute hook for method Approve

#pragma used dll  oppmmmlogging

long  retl  | return value to be checked

if ppmmmlogging.must.do.logging(i.orderType) then
 retl = ppmmmlogging.log.order.approval(i.orderNumber)
 if retl <> 0 then
  dal.set.error.message("ppmmms1234.21")
  |* Logging of approve failed
  return(DALHOOKERROR)
 endif
endif
return(0) | OK
```

Hooks for Repeatable Attributes

When an attribute implementation having dimensions is used in a hook, the interface for the method hooks (before/on/after get/set) will be different. The differences are comparable to attribute hooks.

Example of a before execute hook:

```
if not i.code.isSet(1) or not i.code.isSet(2) then
 io.cancel = true
endif
return(0) | OK
```

Example of an on execute method hook:

```
#pragma used dll oppmmmdll0001
return( ppmmmdll0001.calculate.quantites.and.add.notes(i.item,
```

```
i.item.isSet,
         i.description(1,1), i.description.isSet(1),
         i.description(1,2), i.description.isSet(2),
         io.quantity(1), io.quantity.isSet(1),
         io.quantity(2), io.quantity.isSet(2),
         o.note(1,1), io.note.isSet(1),
         o.note(1,2), io.note.isSet(2)) )
```

Hooks for Batch Methods

Input and Output

Methods having processing order ‘batch’ do not have the method arguments available as parameters. Instead, the whole request is available as an input parameter for the hook. If needed, it can be forwarded to another business object method or DLL.

The following parameters can be used in method hooks for batch methods:

Hook	Request	Response	Result
before execute	i.request (read-only)	-	-
on execute	i.request (read-only)	o.response	o.result
after execute	i.request (read-only)	i.response (read-only)	-

Additionally, just like in method hooks for top-down and bottom-up methods, the before execute method provides an io.cancel (see section Before Execute Method Hook on page 71) and the on execute method provides an io.default parameter (see section On Execute Method Hook on page 73). Note that in this case the io.cancel is on request level, so when setting it to true, the whole request is skipped.

Note however that in fact the before and after execute hooks do not have any added value; you can just as well include all logic you need in the on execute hook.

Using the Request

You can iterate through the instances in the request as follows:

```
#define getFirstInstance(request) xmlFindFirstMatch(sprintf$("<%sRe
quest>.<DataArea>.<%s>",
           "OnEvent", | use the method name here
           "SalesOrder" | use the business object name
           request)
#define getNextInstance(prev_instance) xmlGetRightSibling(prev_instance)

long current.instance
```

```
current.instance = getFirstInstance(i.request)
while current.instance <> 0
 …
 current.instance = getNextInstance(current.instance)
endwhile
```

Alternatively, you can forward the request to another method of the same business object or another business object.

Note: For a class method (such as ‘SubscribeEvent’), no instances are available in the data area.

Using Control Data

Control data is included in i.request and can be added to o.response. You can use the controlling attributes in the before, on and after execute hooks. You can set the controlling attributes in the on execute hook.

Additionally, the helper function getControlAttribute() as described in section Assisting Functions on page 78 can be used.

Using the Response

The o.response output parameter can be used for a response that is built up in the on execute hook, or for a response that is received from another business object method that is invoked from the on execute hook.

Make sure that the contents of o.response matches the definition of the business object method to which the on execute hook is linked.

Do not fill o.response if io.default is set to true.

Error Handling

If you forward the request to (or create a specific request for) another method or business object, you will get a response or result. Use o.result if you get a result. The result will be included in the message details of the final method result.

In that case you must return DALHOOKERROR. It is not needed to use dal.set.error.message().It is not forbidden either.

If an error occurred and you return DALHOOKERROR but without o.result being filled, it is mandatory to use dal.set.error.message(). In that case a result is automatically created based on the set error message(s).

If you fill o.result, set io.default to true and return the OK value (0), the result will be handled as follows:

- If the default process succeeds, the messages from the result are included in the response information area.

- If the default process fails, the messages from the result are included in the method's result.

In a before or after execute hook for a batch method, you cannot return a result XML. However, you can set a DAL error message and return DALHOOKERROR.

If you set a DAL error message in a before/on/after execute hook but return 0 (OK), then the message is regarded as a warning.

Examples

An example of an on execute hook for the OnEvent() method, to illustrate the switch between using your own implementation and using the default behavior:

```
#pragma used dll oppmmmonevent  | my own on event handler

string eventAction(50)

if not getControlAttribute("eventAction", eventAction) then
 dal.set.error.message(…)
 return(DALHOOKERROR)
else
 on case eventAction
 case "myEvent1":
 case "myEvent2":
  | invoke my own event implementation
  return( ppmmmonevent.myEventHandler(i.request, o.response, o.result) )
 default:
  | use the default behaviour
  io.default = true
  o.response = 0
  o.result = 0
  return(0) | OK
 endcase
endif
```

An example of an on execute hook for the OnEvent() method, to illustrate how to forward a request to another business object or method:

```
#pragma used dll oppmmmbl987sb00 | public interface of another business
object

string eventAction(50)

if not getControlAttribute("eventAction", eventAction) then
 dal.set.error.message(…)
 return(DALHOOKERROR)
else
 on case eventAction
 case "approve":
  | use change method of the same business object
  … | at this point the control area in i.request must be changed to match
 the change method
```

```
  return( ppmmmbl123sb00.Change(i.request, o.response, o.result) )
 case "plan":
  | send plan event to another business object
  … | at this point the business object must be renamed in i.request, see
 section 0
  return( ppmmmbl987sb00.OnEvent(i.request, o.response, o.result) )
 default:
  | use the default behaviour
  io.default = true
  o.response = 0
  o.result = 0
  return(0) | OK
 endcase
endif
```

Version Information

The List and Show methods can be implemented using processing order 'batch' and an on execute hook. This can be used for BIIs where all or some components do not map to tables, and for BIDs having multiple implementations (so-called 'chameleons', as described in section Using Multiple Implementations on page 120).

Assisting Functions

Overview

A number of functions is available to help you when implementing BII hooks of libraries that are invoked from hooks.

You can use the following types of assisting functions:

- Public and protected methods, which can be invoked from any hook or application library.
- Helper functions, which are available in hooks only.

These types of functions are described below.

Public and Protected Methods

From a hook you can delegate tasks to another method of the same business object or to another business object by invoking a public or protected business object method. This can be done from BII hooks or from DLLs that are invoked from BII hooks.

For example, you can publish an event from the after execute hook of the Create method. Or you can delegate tasks by invoking a protected business object from the on execute hook of a public business object.

The public methods must be invoked in accordance with the BDE Standard. Regarding protected methods, do not confuse protected methods with other internal or external functions in the protected library. Only protected methods (getters, setters and before/on/after methods and other documented protected methods) must be used.

Helper Functions

The following helper functions are available:

getControlAttribute()

This function can be used in a hook to get the value of any controlling attribute as available in a request. It must not be used in batch implementations.

The control data is available in the before/on/after execute hooks for each method and for each component. Both standard control data and non-standard control data (for specific methods) can be used.

See also section Guidelines on Using or Setting Attribute Implementation Values on page 66.

```
function boolean getControlAttribute(
  const string i.name,
  ref  string o.value)
```

Input:

i.name: name of the controlling attribute.

Output:

- return value: true if the control attribute exists in the request and its value can be retrieved, false otherwise.
- o.value: the value of the control attribute, “” if return value is false.

For example:

```
string batchString(12)
long batchNumber

if not getControlAttribute(“batch”, batchString) then
 batchNumber = 0
else
 batchNumber = lval(batchString)
endif
```

```
do.something(batchNumber)
return(0) | OK
```

getControlAttributeFromRequest()

This function can be used in a hook to get the value of any controlling attribute as available in the request. It must only be used in batch implementations.

```
function boolean getControlAttributeFromRequest(long i.request, const
string i.name, ref string o.value)
```

Input:

- i.request: request provided as input in a before, on or after execute hook.
- i.name: name of the control attribute (case-sensitive).

Output:

- return value: true if the control attribute exists in the request and its value can be retrieved, false otherwise.
- o.value: the value of the control attribute, “” if return value is false.

setControlAttribute()

This function can be used in a hook to set an output control attribute in the response. It must not be used in batch implementations.

```
function boolean setControlAttribute(const string i.name, const string
i.value)
```

Input:

- i.name: name of the control attribute (case-sensitive).
- i.value: value to be set for the control attribute.

Output:

return value: true if successful, false otherwise.

If this function is invoked multiple times, only the last value is used.

setControlAttributeInResponse()

This function can be used in an on execute hook to set an output control attribute in the response. It must only be used in an on execute hook for batch implementations.

```
function boolean setControlAttributeInResponse(long io.response, const
string i.name, const string i.value)
```

Input:

- io.response: request that is to be used as output for an on execute hook.
- i.name: name of the control attribute (case-sensitive).
- i.value: value to be set for the control attribute.

Output:

- return value: true if successful, false otherwise.
- io.response: updated request having the control attribute set (if return value is true).

If the control attribute does not exist in the response it is added. If the control attribute already exists, its value is updated. Consequently, if this function is invoked multiple times for the same response, only the last value is used.

getAttributeValueFromRequest()

This function can be used in a hook to get the value of a data attribute as available in the request. It must only be used in batch implementations. When a request arrives at a batch method implementation, it often must be forwarded to another method and/or business object, depending on the value for one or more attributes. This is also the case when using multiple implementations (so-called ‘chameleon’ implementations), see section Guidelines on Using or Setting Attribute Implementation Values on page 66.

```
function boolean getAttributeValueFromRequest(long i.request, string
  i.path, ref string o.value)
```

Input:

- i.request: request provided as input in a before, on or after execute hook.
- i.path: the complete path to the attribute, starting from the top-level component (case-sensitive).

Output:

- return value: true if successful, false otherwise.
- o.value: the value of the data attribute, “” if return value is false.

Note:

- This function cannot handle qualifier attributes (xml attributes).
- This function will always return the value as a string. And it won’t work for complex content (xml).
- This function can only handle ‘leaf attributes’, so it cannot handle complex XML structures for ‘anyType’ attributes.

- For subcomponents this function will always return the value from the first instance.

Example of a before execute hook for a batch method implementation:

```
boolean retb
string order.status(12)
string amount(16)

retb = getAttributeValueFromRequest(i.request,
    "Order.Header.StatusInformation.Status",
    order.status)
if not retb then
 | for example error if status is mandatory, or use a default order.status

 ...
endif

retb = getAttributeValueFromRequest(i.request,
    "Order.OrderLine.Pricing.Price.Amount",
    amount)
if not retb then
 | for example error if amount is mandatory, or use a default amount
 ...
endif

if tolower$(order.status) <> "approved" or val(amount) < 10000.0 then
 io.cancel = true
endif
return(0) | OK
```

In this case, the second getAttributeValueFromRequest() provides the amount from the first order line in the request.

getRequest()

This function can be used if the request must be forwarded to another business object, but the method is not a batch method. Do not use this for batch methods, because in that case the request is available as an input parameter. Do not change the request or its contents; instead use replaceBdeNameInRequest(), or duplicate the request before changing it.

```
long getRequest()
```

Input: none.

Output:

return value: reference to the request XML, 0 in case of error.

attributeIsSelected()

This function can be used in the before, on and after execute hook of the List and Show methods. It cannot be used if the List/Show is implemented as a batch method.

```
boolean attributeIsSelected(i.componentName,
  i.attributeImplementationName)
```

Input:

- i.componentName: the component containing the attribute. Note that for top-down or bottom-up implementations the component name is superfluous. But for batch implementations it is not. For that reason the component name must be specified.
- i.attributeImplementationName: the attribute implementation name as defined in the BII.

Output:

return value: whether the attribute is selected in the List/Show request.

Example:

```
if attributeIsSelected(Order, businessPartnerName) then
  ...
```

Note: Both the component name and the attribute implementation are not (quoted) strings, so do not invoke this function using attributeIsSelected(“Order”, “orderNumber”), but instead useattributeIsSelected(Order, orderNumber).

Also note that the attribute implementation name may differ from the public attribute name, especially if attribute grouping is used. For example, you can use:

attributeIsSelected(Order, Header_IDs_orderNumber).

The return value is true if the attribute implementation is selected, and false if not. In case of derived protected attributes (having an on set hook), selection is also derived from the selection of the public attribute.

For example, for an internalOrderNumber which is derived from the public orderNumber, attributeIsSelected(Order, internalOrderNumber) will return true if attributeIsSelected(Order, orderNumber) returns true.

replaceBdeNameInRequest()

This function replaces the business object name wherever it occurs in the request. It can be used when forwarding a request from one business object to another business object. The old business object name will be the name of the current business object (that contains the hook). If the request belongs to another business object, the method will fail.

The business object name is replaced for example in the selection (when selecting attributes or ‘*’ for the top-level component), in the filter (when filtering on attributes from the top-level component), in the children of the DataArea node if the data area contains any instances, and in the eventEntity controlling attribute for the OnEvent method.

Limitation: only the data area and the standard controlling attributes are handled (so specific controlling attributes are not renamed if they contain the business object name).

```
long replaceBdeNameInRequest(
              long        i.request,
  const       string      i.newBdeName,
  ref         long        o.newRequest)
```

Input:

- i.request: request xml
- i.newBdeName: new business object name to be used

Output:

- o.newRequest: duplicate of i.request, where the business object name is replaced (note: this xml must be cleaned up after use!)
- return value: 0 (OK) or an error value.

replaceBdeNameInResponse()

This function replaces the business object name wherever it occurs in the response. It can be used after forwarding a request from one business object to another business object. The new business object name will be the name of the current business object (that contains the hook).

Limitation: only the data area and the standard controlling attributes are handled (so specific controlling attributes are not renamed if they contain the business object name).

```
long replaceBdeNameInResponse(
              long        i.request,
  const       string      i.oldBdeName,
  ref         long        o.newResponse)
```

Input:

- i.response: response xml
- i.oldBdeName: old business object name to be replaced

Output:

- o.newResponse: duplicate of i.response, where the business object name is replaced (note: this xml must be cleaned up use!)
- return value: 0 (OK) or an error value.

removeStandardEventsFromRequest()

This method cannot be used for BOD-type business objects.

This function removes the standard event actions from a SubscribeEvent request.

```
long removeStandardEventsFromRequest(
             long         io.request,
  ref        boolean      o.anyEventActionLeft)
```

Input:

io.request: SubscribeEvent request xml

Output:

- io.request: request after removing eventAction elements for standard events (create, change, delete)
- o.anyEventActionLeft: true: at least one (specific) event action is remaining in the control area; false: no event actions are listed anymore. This output parameter is needed, because when forwarding a request having no event actions specified, this is interpreted as a request for all standard events!
- return value: 0 (OK) or an error value.

suppressStandardPublisher()

This method cannot be used for BOD-type business objects.

In the implementation for the SubscribeEvent method, the business object developer can indicate if standard events originate from the application instead of from the audit trail. This is done through a helper method called suppressStandardPublisher(), which can be used in the on execute hook for the SubscribeEvent method.

The suppressStandardPublisher() can be called without parameters or with one or more strings containing standard event actions (“create”, “change” or “delete”). The sequence of the arguments does not matter.

For example:

- When specifying suppressStandardPublisher(), all events must be sent from the LN application, and the publisher will not pick up standard events from the audit trail.
- When specifying suppressStandardPublisher(“change”, “delete”), events having eventAction ‘change’ or ‘delete’ must be sent from the LN application, and a standard publisher process will only read from the audit trail if the subscription request contains the ‘create’ eventAction.

Specifying suppressStandardPublisher(“create”, “change”, “delete”) has the same effect as specifying suppressStandardPublisher().

Using suppressStandardPublisher() in the on execute hook for SubscribeEvent is only useful if io.default is set to true (see section On Execute Method Hook on page 73). Otherwise the default process is not executed, so no publisher is started either.

Example of an on execute hook to be used when the standard publisher process must only handle ‘create’ and ‘delete’ events, while ‘change’ events are generated from the application:

```
suppressStandardPublisher(“change”)
io.default = true
return(0) | OK
```

This hook can be used if the top-level component of the business object is mapped to an LN database table, but it has subcomponents that are not directly related to tables.

Chapter 4: Method-Specific Guidelines

This chapter discusses guidelines to take into account when implementing standard methods.

General

Method Dependencies

The dependencies in the following table must be taken into account.

If you implement	You must also implement	Notes
SubscribeEvent	UnsubscribeEvent	
SubscribeEvent	PublishEvent	
SubscribeEvent	Show	Unless the SubscribeEvent has an on execute hook and no default behavior.
SubscribeList	List	You don’t have to implement the SubscribeList; it is available automatically based on the List method
SubscribeList	PublishEvent	
Change	Create, Delete	Unless the BID has no subcomponents
Delete	Show, Change	Unless the BID has no subcomponents

For BODs, the (protected) PublishEvent method must be available, but no SubscribeEvent and UnsubscribeEvent methods are needed.

Method Arguments

The following table specifies the requirements for the method argument scope per method.

Method	Method argument implementations
List	Scope ‘out’, except for unrelated attribute implementations that represent the identifiers of parent components, which must have scope ‘inout’.
Show	Scope ‘out’, except for identifying attribute implementations, which must have scope ‘inout’
Create	Scope ‘in’ or ‘inout’, except for attributes for which the value is generated in LN, which must have scope ‘out’
Change	Scope ‘in’ or ‘inout’, except for identifying attributes, which must have scope ‘in’.
Delete	Scope ‘in’; only method argument implementations exist for identifying attributes.
SubscribeEvent	No arguments
UnsubscribeEvent	No arguments
PublishEvent	Scope ‘in’
OnEvent	Scope ‘in’
SubscribeList	No arguments
SupportsProcessingScope	No implementation needed
SupportsReferentialIntegrity	No arguments
CreateRef	Scope ‘in’; only method argument implementations exist for identifying attributes.
DeleteRef	Scope ‘in’; only method argument implementations exist for identifying attributes.

Processing Order

The processing order for standard methods must be as follows:

Method	Processing Order
List	topDown or batch (batch only if the method has an on execute hook and no default behavior) (*)
Show	topDown or batch (batch only if the method has an on execute hook and no default behavior) (*)
Create	topDown or batch (batch only if the method has an on execute hook and no default behavior)
Change	topDown or batch (batch only if the method has an on execute hook and no default behavior)

Method	Processing Order
Delete	bottomUp or batch (batch only if the method has an on execute hook and no default behavior)
SubscribeEvent	batch
UnsubscribeEvent	batch
PublishEvent	batch
OnEvent	batch
SubscribeList	batch
SupportsProcessingScope	No implementation needed
SupportsReferentialIntegrity	batch
CreateRef	topDown or batch (batch only if the method has an on execute hook and no default behavior)
DeleteRef	bottomUp or batch (batch only if the method has an on execute hook and no default behavior)

List and Show Methods

Overview

The List and Show methods can be customized.

The most important differences with older versions are:

- Method hooks (before execute, on execute, after execute) can now be used for the List and Show methods.
- The new implementation allows for more complex mappings. In the BII you can use non-root tables that are not linked to the root table through the identifiers of both tables.
- The filtering is enhanced. You can use complex filters across components, such as Order.status = 'Open' and (Line.item = 'ABC' or Line.Item = 'DEF') or Order.deliveryDate is empty.

Note: The changes do not necessarily result in a better performance for List and Show. In a number of cases, the increased flexibility will even result in longer response times.

Complex mappings

For the List and Show methods, the relation between root and non-root tables need not be defined through the table identifiers. Relations between components must always be at the 'inside'.

Implementing Hooks for List and Show

Introduction

Hooks can be used for the List and Show methods. To a large extent, the flow for List and Show is comparable to the flow for other top-down methods such as Create or Change. However, some differences exist, because the number of component instances processed for Create and Change depend on the request contents, while the number of component instances processed for List and Show depend on the data that is read from the database.

Consequently, in the before execute hook for Create and Change you are sure a specific instance is being processed. In case of List and Show the next instance has not been read yet, and may not even exist. So for List and Show the before, on and after method hooks are invoked per instance, but these functions are executed 'once too often'. This is because the last invocation of the execute function will fetch nothing. In the before execute and on execute hooks for Create and Change, all attribute values will be input, while in case of List and Show most attribute values will be output only.

Note: A List or Show method cannot be bottom-up. They can only be top-down or batch. If List and/or Show are implemented as batch methods, the interfaces differ.

The implementation for Show is comparable to the implementation of List. The main difference is that the input differs for the top-level component. Actually, we can say that the Show is a special type of List, having a filter on the identifiers for the top-level component. Although technically the Show method is not implemented this way.

In case of a batch implementation for the List method, the on execute hook must handle the filter, selection, iterator and 'maxNumberOfObjects'. In case of a top-down implementation, the iterator and 'maxNumberOfObjects' are handled in the public layer. You do not need to take these into account in the hook code.

Regarding selection and filtering see the following section.

Selected Attributes

Which attributes are selected is not communicated through input parameters for the hooks. Instead, the following helper function is available:

```
boolean attributeIsSelected(i.componentName, i.attributeImplementationName)
```

This helper function can be used in the before, on and after execute hook. It cannot be used if the List/Show is implemented as a batch method.

Input is the attribute implementation name as defined in the BII. and the component name as defined in the BII. Note that for top-down or bottom-up implementations the component name is superfluous. But for batch implementations it isn't. For that reason the component name must be specified.

Example:

```
if attributeIsSelected(Order, businessPartnerName) then
 ...
```

Note: Both the component name and the attribute implementation are not (quoted) strings, do not invoke this function using attributeIsSelected("Order", "orderNumber"), but use attributeIsSelected(Order, orderNumber) instead.

Also note that the attribute implementation name may differ from the public attribute name, especially if attribute grouping is used. For example, you can use: attributeIsSelected(Order, Header_IDs_orderNumber).

The return value is true if the attribute implementation is selected, and false if not. In case of derived protected attributes (having an on set hook), selection is also derived from the selection of the public attribute. For example, for an internalOrderNumber which is derived from the public orderNumber, attributeIsSelected(Order, internalOrderNumber) will return true if attributeIsSelected(Order, orderNumber) returns true

Filtering

The filter used in the List or Show request is available as input (i.filter) for the before execute hook and the on execute hook. The filter is an xml structure containing a standard filter as used for the List and Show methods. If no filtering is applied, the filter may be 0 or contain an empty filter node (<Filter/>).

In case of a batch implementation, the hook must take care that filtering is done. If the List or Show request is delegated to another business object, that business object will take care of that.

In case of a top-down implementation, the filter does not have to be used in the on execute hook. The standard generated logic will apply ‘postfiltering’ if needed. Postfiltering is needed in two situations. (1) If one of the components being filtered has an on execute hook. (2) If one of the components being filtered does not have an on execute hook itself, but has a parent that has an on execute hook. That means the filter is applied after building the response message. However, the filter can be used in the on execute hook to optimize performance, by reducing the number of component instances being returned.

Example

For top-level and child components the input and output method arguments for the method hooks are described here using examples.

In these examples we assume that the following structure exists:

- Component Order
  - Attribute orderNumber
  - Group Customer
    - Attribute name
  - Component Line

- Attribute lineNumber
- Attribute item

Let's furthermore assume that the orderNumber is a calculated attribute and internally two attribute implementations exist: orno1 and orno2

The following attribute implementations are listed as method argument implementations:

Component	List	Show
Order	Input: none Output: orderNumber, Customer_name	Input/output: orderNumber Output: Customer_name
Line	Input: orno1, orno2 (*) Output: lineNumber, item	Input: orno1, orno2 (*) Output: lineNumber, item

(*) Do not use orderNumber here!

If two orders exist, where the first one has two lines and the second one has no lines, the hooks will be invoked as follows:

Component - Hook	Notes
Order - Before execute	
Order - On execute	Fetches first order
Line - Before execute	
Line - On execute	Fetches first line
Line - After execute	
Line - Before execute	
Line - On execute	Fetches second line
Line - After execute	
Line - Before execute	
Line - On execute	Fetches nothing
Line - After execute	
Order - After execute	
Order - Before execute	
Order - On execute	Fetches second order
Line - Before execute	
Line - On execute	Fetches nothing
Line - After execute	
Order - After execute	

Component - Hook	Notes
Order - Before execute	
Order - On execute	Fetches nothing
Order - After execute	

The 'after execute' hook is invoked even though nothing is fetched, to enable cleaning up any data or state that was initialized in the 'before execute' or 'on execute' hook.

Details on the before, on and after execute hook are described in the following section.

Note: A subset of the hooks can be implemented. Implementing an on execute hook for one component doesn't mean it has to be done for the other components as well. And implementing an on execute hook doesn't always mean you also must implement a before execute hook.

Before Execute Hook

This hook is executed before each fetch (even if nothing will be fetched!).

Before Execute	List	Show
Input for top-level component	i.filter	i.filter, Public identifiers: i.orderNumber, i.orderNumber.isSet
Input for child component	Internal identifiers from parent component: i.orno1, i.orno2, orno1.isSet, i.orno2.isSet	
Output	o.cancel	

Note: for the List/Show methods the 'isSet' input parameters are not really needed, because the input identifiers will always be set.

In case of a subcomponent, whether lines for a new order must be retrieved can be detected through the input arguments for the order number. So no specific input parameter exists to indicate this.

On Execute Hook

This hook is executed on each fetch.

On Execute	List	Show
Input for top-level component	i.filter	i.filter, Public identifiers: i.orderNumber, i.orderNumber.isSet

On Execute	List Show
Input for child component	Internal identifiers from parent component: i.orno1, i.orno2, orno1.isSet, i.orno2.isSet
Output for top-level component	o.anythingFetched, All attribute arguments: o.orderNumber, o.orno1, o.orno2, o.Customer_name, o.orderNumber.isSet, o.orno1.isSet, o.orno2.isSet, o.Customer_name.isSet
Output for child component	o.anythingFetched, All attribute arguments: o.orno1, o.orno2, o.lineNumber, o.item, o.orno1.isSet, o.orno2.isSet, o.lineNumber.isSet, o.item.isSet,

The on execute hook must set o.anythingFetched to the correct value. If it is set to true then the data will be retrieved and an object or component instance is added to the List/Show response. If it is set to false then in case of a subcomponent the process will continue reading the next parent, and in case of the top-level component, the execution for the method will be finalized

Note: For List/Show we do not support io.default. If an on execute hook is defined, the default behaviour cannot be used. The reason is that the default behaviour requires initialization (building a query, parsing and executing it) that cannot be done anymore at the moment of 'on execute'. Additionally, we cannot switch between default behaviour or not per instance, because of the state that is used while reading the database.

After Execute Hook

This hook is executed after each fetch and also after the last execute, when nothing is fetched.

After Execute	List Show
Input for top-level component	i.anythingFetched, i.errorOccurred, All attribute arguments: i.orderNumber, i.orno1, i.orno2, i.Customer_name, i.orderNumber.isSet, i.orno1.isSet, i.orno2.isSet, i.Customer_name.isSet
Input for child component	i.anythingFetched, i.errorOccurred, All attribute arguments: i.orno1, i.orno2, i.lineNumber, i.item, i.orno1.isSet, i.orno2.isSet, i.lineNumber.isSet, i.item.isSet
Output	None

The i.anythingFetched will indicate whether the last (on) execute returned any data. Compare the o.anythingFetched parameter of the on execute hook. The i.errorOccurred will indicate whether the last (on) execute resulted in an error. If so, i.anythingFetched will be false. The after execute hook is

invoked in case of error to enable any cleaning up of state that was set in the before or on execute hook.

Impact on Other Methods

The reference methods will need to be changed if a component is not mapped to a table (so the List/Show methods will have an on execute hook defined). In that case the reference methods will also need an on execute hook.

Traversal by Association

When developing business interfaces, associations from one business object to another can be defined. Associations are modeled in the BID using the LN Studio. An association contains an association name and the name of the associated business object, as well as a relation from attributes of the business object to all identifiers of the associated business object or component. Such a relation is one-to-one or many-to-one.

Multiple associations can exist from one business object or one component to another. For example, an Order business object or an OrderLine component may contain multiple currency codes or unit codes.

Associations can be used to explicitly add an associated attribute to a BID. For example, adding an itemDescription attribute in an order line. This functionality was available already in the Integration 6.1 release on top of Enterprise Server 8.3.

For traversal by association, the developer does not have to add the associated attributes to the BID or BII. Instead, they can be selected at runtime.

For example, if an itemCode is available on an OrderLine component and an association called ‘OrderedItem’ to the Item business object is defined, then a client application can include the following in a List or Show selection:

```
<Selection>
<selectionAttribute>OrderLine.OrderedItem.Item.itemDescription</selectionAttribute>
<selectionAttribute>OrderLine.OrderedItem.Item.itemGroup</selectionAttribute>
<selectionAttribute>OrderLine.OrderedItem.Item.cataloguePrice</selectionAttribute>
</Selection>
```

Note: To make use of traversal by association, both the invoked business object and the associated business object must be generated from LN Studio. Additionally, the generators for BDE proxies (such as Java and .Net) may not support the user of traversal by association at runtime.

In LN Studio you can model association relationships in the BII without a corresponding association in the BID. In that case traversal by association cannot be used. You can however use the association in an attribute implementation.

Traversal by association is available with List, Show, SubscribeList and SubscribeEvent methods. In case of SubscribeEvent, events occurring on the associated business object are by default not detected and published for the subscribed business object.

Note: Traversal by association can only be used for selections; it cannot be used in filters.

Additionally, the business object runtime only supports associations if:

- An association does not have the same name as an attribute, attribute group or subcomponent.
- The association is defined from a single component, using attributes from that component only. In the LN Studio, you cannot model an association from multiple components. For example, if a PurchaseOrderLine contains a ‘relatedSalesOrderLine’ attribute and the PurchaseOrder (not the PurchaseOrderLine component) contains a ‘relatedSalesOrder’ attribute, you cannot use these to associate to the SalesOrderLine.
- The associated business object (or component) is referred to through all its public identifiers. For example, through the itemCode for an association to an Item business object, or through the salesOrderNumber and lineNumber for an association to the SalesOrderLine component of a SalesOrder business object. This is enforced by the LN Studio user interface.

Referential Integrity Methods

When using the Business Object Repository, referential integrity tables were automatically generated when converting to runtime. This is not the case anymore from the LN Studio. This section describes how to set up referential integrity methods.

Note: According to the DeleteRef specifications in the BDE standard, all references for a specific referenceID can be deleted by using no or an empty data area. This functionality is currently unsupported in LN. Instead, delete all references by explicitly specifying the business object instances (or component instances) for which a reference was defined. In other words, the data area used for the CreateRef must also be used for the corresponding DeleteRef.

Let’s for example assume a BII exists for LN. Two components are defined, called ‘Order’ and ‘OrderLine’.

The mapping to tables and columns is as follows:

Component	Attribute / Attribute Implementation	Table	Column
Order	orderNumber	ppmmm001	orno
OrderLine	orderNumber (not a public attribute)	ppmmm002	orno
OrderLine	lineNumber	ppmmm002	lino

In the following section we will show how to implement the CreateRef and DeleteRef methods for both the Order and the OrderLine.

Table Definitions and DALs

In LN, create the following two tables:

- ppmmm901
- ppmmm902

The tables must be created in a VRC that is equal to, or is a predecessor of, the VRC that will contain the generated libraries for the LN business object implementation.

The tables must have the following definition:

Table ppmmm901 – External References for ppmmm001.

Column	Domain	Notes
orno	Same as ppmmm001.orno	mandatory
ref_rkey	A string(40) domain such as tcm-cs.st40	mandatory
cmba	n/a	combined (orno, ref_rkey), primary key
cmbb	n/a	combined (orno), reference to ppm-mm001.orno, mandatory reference using lookup check

Note:

Strictly spoken the combined columns are not needed in this case, but they are used to keep the definition consistent for all referential integrity tables.

A mandatory reference to the component's root table must always be defined; otherwise the CreateRef will not have any effect, because the referential integrity can not be checked.

Table ppmmm902 – External References for ppmmm002:

Col-umn	Domain	Notes
orno	Same as ppmmm002.orno	mandatory, part of primary key
lino	Same as ppmmm002.orno	mandatory, part of primary key
ref_rkey	A string(40) domain such as tcm-cs.st40	mandatory, part of primary key
cmba	n/a	combined (orno, lino, ref_rkey), primary key
cmbb	n/a	combined (orno, lino), reference to ppmmm002.cm-ba (orno, lino), mandatory reference using lookup check

Convert the table definitions to runtime. Create the tables in the companies as needed and create a DAL (Data Access Layer) script for each table.

Note: The column names for ref_rkey and the identifiers must be as specified earlier in the tables. In other words, the Reference Key column must always be called ‘ref_rkey’, and the identifier columns must always have the same code as columns for the component’s root table.

This implies that we cannot define reference tables for BIIs having no mapping to (other) tables. In theory we can, but the referential integrity will not be checked if there is no reference in the data model from the reference table to the business object component’s root table.

BII

In the LN Studio, import the tables. Link them to the corresponding components and set the ‘Is Referential Integrity Table’ property to ‘true’. The ‘Is Root Table’ property must be ‘false’. There is no need to explicitly define a relation between the component’s root table and its referential integrity table. Additionally, no attribute implementations must be mapped to referential integrity table columns.

Chapter 5: Publishing and Receiving Events for BDE only

Introduction

Overview

When subscribing to one or more standard events (create, change, delete) in LN, a publisher is started that will detect the events based on the audit trail and will publish the corresponding event messages.

When subscribing to specific events, the subscription is stored in LN and the publisher will be running, but in fact the subscription is passive. The publisher will not pick up anything until an event triggered from the application.

Events can be published either based on the audit trail or from the LN application.

The following methods exist to initiate an event from the LN application:

- Using the standard PublishEvent method. In that case the application is responsible for building the request (XML) in accordance with the BDE interface definition. See following section “PublishEvent”.
- Using the ShowAndPublishEvent method. In this case the application can initiate the event at business object or component level simply by specifying the values for the identifying table columns. The business object runtime will take care of collecting the required data, building the request XML and invoking the PublishEvent method to do the actual publishing. For details, see following section “ShowAndPublishEvent”.
- Using the ShowAndPublishStandardEvent method. This method is available to publish standard events from the application in addition to (or instead of) publishing them based on the audit trail. This method is comparable to ShowAndPublishEvent, but it cannot be used for application-specific events, but for standard events (create, change, or delete) only. Note that also the PublishEvent method can be used to publish standard events. For details, see following section “ShowAndPublishStandardEvent”.

In theory, the PublishEvent, ShowAndPublishEvent or ShowAndPublishStandardEvent can be invoked from any script or library in the application. Logical places to implement the invocation are for example the data access layer (for events that are related to database changes or execution of a business method) or the user interface script (for events that are linked to a specific form command, for example).

When LN acts as an event consumer, incoming events are processed through the OnEvent method.

Transactional Event Publishing

Publishing from the LN application is transactional. This means the publishing is done within a database transaction. The actual publishing is postponed until the end of the transaction. If the transaction is committed, the publishing is done. If the transaction is aborted, the publishing is not done.

When PublishEvent or ShowAndPublish(Standard)Event is invoked from the application, first the subscription is checked. If a subscription exists for the specified business object and

eventAction, then the request is stored in a queue. The enqueuing will only be effective if the transaction is committed by the application. In case of an abort, nothing will be published.

A publisher process is not only running in case of a SubscribeEvent on standard event actions, but for all event actions (also when the standard publishing process is suppressed in an on execute hook for SubscribeEvent). The publisher does not only check for standard events as logged in the audit trail, but also for application events from the queue. The events from both sources are sequenced correctly. Technically, the audit trail facility is reused for the application events, so the time each application event gets a commit time and a transaction ID.

This approach has a number of advantages:

1. When publishing inside a transaction, the event will not be published if the application's database transaction is aborted.
2. All events (from PublishEvent, from ShowAndPublish(Standard)Events and from the standard audit trail entries) are sequenced correctly.
3. The performance impact for the application is limited, because the aggregation and publishing of events is done 'offline'.

Note: This means that PublishEvent PublishEvent, ShowAndPublishEvent and ShowAndPublishStandardEvent must always be invoked from the application inside a database transaction. So a retry point must be set before publishing and a commit or abort must be done afterwards. This must be the same application transaction that actually initiates the event. For example, if an Order row is changed and an event is published because of that, these must be in the same transaction scope. This way, an event is only published if the corresponding database change succeeds.

Identifier Mapping

For both application-based publishing and audit-based publishing, it is important that the component and table identifiers are mapped correctly.

If a table mapping is defined for a component, then the component identifiers must be mapped to the root table identifiers and vice versa.

For application-based event publishing using one of the ShowAndPublish methods, if a component is not mapped to a table, but calculated identifiers exist then the component identifiers must be mapped to the internal identifiers and vice versa.

Testing

For testing and troubleshooting an event publishing implementation, refer to section Testing on page 41.

Audit-Based Publishing

By default, standard audit-based publishing is available for BDE implementations. This means that database actions on relevant table columns are translated to Create, Change and Delete events on the business object.

Impact on the BII

In case standard events from LN are published audit-based, you must take the following into account when modelling a BII.

Changes on business object instances in LN are only detected if they correspond to changes on one or more LN table columns. This means that to use the standard event publishing, at least one business object component must be mapped to a database table.

Regarding calculated attributes please note the following.

Calculations as modeled in the BII are used for standard event publishing. The assumption is that the calculation is modeled completely. So if we ‘set’ the relevant attribute implementations that are mapped one-to-one on table columns, and we subsequently ‘get’ the public attribute that is mapped to those attribute implementations, we must get the right value.

Consequently, change events may not be detected if 'work-arounds' using hand-crafted code is implemented, such as:

- Coding calculation logic in the wrong place in the BII hooks.
- Using values from other table columns or attributes that are not defined as ‘used attributes’.
- Using an attribute that is mapped to nothing, but in the implementation code does select data from the business object's table(s). For example, for an order line attribute, program an SQL query on the order header table to calculate the value, or invoke an existing library function that does something like that.

Application Events

The audit-based publishing will only publish standard events (create, change or delete). In case the PublishEvent method is available and this method must be able to publish application (business object-specific) events, then the generation of those events must be implemented in the application. This is explained in the following sections.

There may be other reasons for not using the standard audit-based publishing. For example, if the table mapping is incomplete (on get, on set or on execute hooks are used instead), or if you need to publish more events or less events than the number of database actions. The publishing of standard events from the application is also discussed in the following sections.

PublishEvent

Public Method

The PublishEvent method is available in the public library. The method name is <prefix>.PublishEvent.

The input/output for this method:

- Input request (long): xml containing the PublishEvent request. Include the control attributes and data area in accordance with the BDE standard. In the data are, include the complete business object instance where possible. From the application point of view it may be sufficient to only include an identifier (such as an order number). But that means that no filtering can be applied on any other attributes or components.
- Output response (long): xml containing the response. Because the error handling is done inside the PublishEvent method, it will only contain a single ‘PublishEventResponse’ node. It must be cleaned up using xmlDelete() after invoking the method.
- Output result (long): xml containing the result. Because the error handling is done inside the PublishEvent method, it will always be empty.
- The return value will always be 0.

Example

If the public library of a business object is ppmmmbl999sb00 then the interface is as follows:

```
long ppmmm.bl999sb00.PublishEvent(long i.request, long o.response, long
o.result)
```

Specifications

The following happens when invoking the PublishEvent() method:

1 If no subscription exists for the business object then nothing is done. Otherwise the following steps are executed.

2 The request is enqueued. After committing the transaction, the request is picked up by a publisher process, which publishes the event, if it matches the subscription filter.

Note:

- PublishEvent can be used for both standard events (create, change or delete) and application-specific events.

  See also section How to Implement this in the Application on page 107.

- In the event being published, actionType elements can be used. Any actionType values can be used. However, in case of standard actionTypes such as create, change and delete, the developer is responsible for using action types that are valid and match the specified eventAction.
- The method must be called from within a database transaction.

  See section Introduction on page 99.

- After invoking the method clean up this request xml and response xml using xmlDelete().

Transactional Event Publishing

If an on execute hook is linked to the PublishEvent, this hook is executed immediately within the user transaction (without enqueuing first). If io.default is set to true, the event is enqueued after executing the hook. The hook is not executed twice for that event.

Background:

It is not possible to first check the subscription and enqueue, because we don’t know what is coded in the hook. For example, the hook may write to a table, thus requiring a database transaction context (which the publisher process does not have). Or the hook may change the contents of the event before executing the default process, so we cannot check the subscription before invoking the hook. Or the hook may contain subscription-independent actions, so again we cannot check the subscription before invoking the hook.

This implies that a PublishEvent hook is not always executed at the same moment:

- If the application invokes PublishEvent directly, the hook is invoked inside the application transaction. (No queuing is done.)
- If the application invokes ShowAndPublish(Standard)Event, the event is first queued, so the PublishEvent is invoked asynchronously, outside the application transaction and application context.

Events from ShowAndPublishEvent and ShowAndPublishStandardEvent are also published through PublishEvent. However, in that case the publishing will immediately be done; it will not be enqueued and dequeued a second time.

ShowAndPublishEvent

Note: Using the ShowAndPublishEvent method it is not possible to generate standard events (create, change, delete). Use ShowAndPublishStandardEvent on page 106 instead.

Protected Method

To publish application events, a method is available in the protected library.

The method name is <prefix>.<component>.ShowAndPublishEvent().

The first argument of this method is a string indicating the event action. For example: “approve” or “cancel”. The other arguments (at least one) contain the values for the attribute implementations that represent the root table’s key columns.

Example

Let’s assume that an Order business object exists. The protected layer is implemented in library ppmmmbl999st00. The root component maps to table ppmmm999.

Primary key of table ppmmm999 is otyp, orno. The business object has two attribute implementations for these columns: orderType and orderNumber. The public identifier is orderID, which is calculated from orderType and orderNumber.

In that case the method will be:

```
ppmmm.bl999st00.Order.ShowAndPublishEvent(
  const      string        i.eventAction,
  domain     ppmmm.otyp    i.orderType,
  domain     ppmmm.orno    i.orderNumber
             [boolean      i.instanceAvailable,
  const      string        i.actionType])
```

Input:

- The i.eventAction can be any string (as long as it matches the BDE standard for event actions), except one of the standard event actions (“create’, “change”, “delete”). It must always be filled.
- The parameter(s) after i.eventAction are the internal identifiers for the object or component instance to be published. In this example, two parameters are used. The implementation of this method will determine the value for the orderID based on the second and third input parameters and then it will build and publish the corresponding event.
- i.instanceAvailable: optional, true if unspecified; if false, only identifiers are included in the event message and the Show method is unused; this must only be set to false if the instance is unavailable so the Show will fail!
- i.actionType: if empty (or unspecified) no actionTypes are set, if filled this action type will be used for the component instance and all other component instances (parents/children) will get action type ‘unchanged’.

If the method is invoked as follows:

```
ppmmm.bl999st00.Order.ShowAndPublishEvent(
   “cancel”,
       ppmmm.otyp.purchase, “123”)
```

then the following event is published, for example:

```
<EventMessage>
 <ControlArea>
  <eventEntity>Order</eventEntity>
  <eventAction>cancel</eventAction>
  <eventTimeStamp>2007-01-16T10:56:38Z</eventTimeStamp>
  <eventSupplier>LN Company 590</eventSupplier>
 </ControlArea>
 <DataArea>
  <Order>
   <orderID>PUR_00123</orderID>
   <orderDate>2007-01-10T10:56:38Z</ orderDate >
   <Line>
    <orderLineNumber>1</orderLineNumber>
    <item>100</item>
    <quantity>12</quantity>
    <price>9.99</price>
    <orderedQuantityValue>100</orderedQuantityValue>
   </Line>
   <Line>
    <orderLineNumber>2</orderLineNumber>
    <item>200</item>
    <quantity>1</quantity>
    <price>39.95</price>
   </Line>
  </Order>
 </DataArea>
</EventMessage>
```

Specifications

The following happens when invoking the ShowAndPublishEvent() method:

1. If no subscription exists for the business object then nothing is done. Otherwise the following steps are executed.
2. The request is enqueued. After committing the transaction, the request is picked up by a publisher process, which executes the next steps.
3. The public identifying attribute value(s) are determined based on the protected identifier.
4. The Show method is invoked to retrieve the object instance. The filter and selection as specified in the subscription from the event consumer are used. If the object instance does not match the filter then nothing is published.
5. The PublishEvent() method is invoked to do the actual publishing, depending on the existing subscription.

Note:

- Error handling: The method will not fail and consequently will not return an error. Instead, if an error occurred the message is published to the error handler that is linked to the destination. For example, if the specified identifying values can not be translated to a valid public identifier, or if the Show invocation failed. If the environment (or package combination) does not contain an Adapter

for LN or if it contains a version that does not support event publishing then a log entry is written in the $BSE/log directory.

- The method must be called from within a database transaction.

  See section Introduction on page 99.

- Every business object that has a PublishEvent() method also has a ShowAndPublishEvent() method. The ShowAndPublishEvent() must not be modeled in the LN Studio and cannot be customized. You cannot define method hooks, for example.

Using ShowAndPublishEvent for Subcomponents

A ShowAndPublishEvent() method can also be used for subcomponents. For example:

```
ppmmm.bl999st00.OrderLine.ShowAndPublishEvent(
  const      string          i.eventAction,
  domain     ppmmm.otyp      i.orderType,
  domain     ppmmm.orno      i.orderNumber,
  domain     ppmmm.line      i.lineNumber)
```

When invoking ShowAndPublishEvent() for a subcomponent, then the following component instances are included in the event message: the specified component instance, all its children, grand children etc., and its parent, grand parent etc. Siblings of the specified components and of its parents are not included.

For example, for a business object consisting of three levels:

- ShowAndPublishEvent on order X: include orders X and all order lines of X and all sub lines of those order lines (depending on selection and filter, of course).
- ShowAndPublishEvent on order X line 1: include order X and order line 1 and all sub lines order line 1 (depending on selection and filter, of course). So order line 2 etc. are not included.
- ShowAndPublishEvent on order X line 1, sub line 3: include order X and order line 1 and sub line 3 (depending on selection and filter, of course). So order line 2 etc. are not included and also sub line 1, 2 and 4 of order line 1 are not included.

ShowAndPublishStandardEvent

Introduction

It is possible to publish standard events (create, change, or delete) from the LN application.

Normally, when receiving a SubscribeEvent request that includes one or more standard event actions (create, change, or delete), auditing is switched on for the business object's database tables. The audit

trail is automatically populated, so standard events need not be published from the LN application. A publisher will be running to pick up the database changes and publish the corresponding events.

Note that in some cases change events cannot always be detected from the audit trail. For example, if the business object, or some of its subcomponents, are not directly related to LN tables. In such cases, the use of a publisher based on the audit trail for the business object tables can be suppressed. This can be configured in the BII (more specifically, in the on execute hook for the SubscribeEvent method).

Some examples:

- If the BII specifies that no standard publisher must be started for any standard events, the subscription is simply stored and the publisher will wait only for application events. to arrive. Consequently, all standard events must be generated from the LN application.
- If the BII specifies that no standard publisher must be started for the standard ‘change’ event, and a subscription arrives for ‘create’, ‘change’ and ‘delete’ events, then a standard publisher will be started only for the ‘create’ and ‘delete’ events, and the ‘change’ event must be published from the LN application.

This chapter explains how this must be configured (programmed). How to Model this in the BII on page 107 explains how to model the business interface implementation in the LN Studio. How to Implement this in the Application on page 107 explains how to implement the publishing of standard events in the application.

How to Model this in the BII

Specifications

In the implementation for the SubscribeEvent method, the business object developer can indicate if standard events originate from the application instead of from the audit trail. This is done through a helper method called suppressStandardPublisher(), which can be used in the on execute hook for the SubscribeEvent method.

For details, refer to section Assisting Functions on page 78.

How to Implement this in the Application

Using PublishEvent

When publishing standard events through PublishEvent, make sure that the event message matches the BDE standards. For example:

- For eventAction create: all component instances are included, and for each component, all attributes are included
- For eventAction change: changed (or created or deleted) component instances are included in the message, including their parents. For each component, all attributes are included. If a component is created, deleted, or unchanged, the actionType attribute is set accordingly.
- For eventAction delete: only the top-level component is included, and it contains only identifying attributes.

Regarding filtering, see Filtering Notes on page 109.

Using ShowAndPublishStandardEvent

The interface for publishing standard events from the application is the protected ShowAndPublishStandardEvent() method. This method is comparable to the ShowAndPublishEvent() method (see former section, “ShowAndPublishEvent”), except that the event action is not specified, but the action type at component level is specified.

The interface is as follows:

```
ppmmm.bl999st00.OrderLine.ShowAndPublishStandardEvent(
  const      string          i.actionType,
  domain     ppmmm.otyp      i.orderType,
  domain     ppmmm.orno      i.orderNumber,
  domain     ppmmm.line      i.lineNumber)
```

The action type is the action for the component. It must always be filled; allowed values are “create”, “change” and “delete”. The event action is not specified explicitly. Instead it is derived from the action type. In this case (because OrderLine is a subcomponent) action type create, change or delete will result in an event having eventAction ‘change'. For a top-level component, the event action will automatically be the same as the action type.

Note: Be aware that a ‘create’ or ‘delete’ on a non-root table row must be published as a ‘change’ on the component. An action on a non-root table does not create or delete the component instance, because the corresponding row on the root table is not created or deleted.

In case of a delete action, only the specified component instance and its parent(s) are published. No deleted child components are published. (We can try to read and add children using action type 'delete'; but that is not logical. It will mean that the event is inconsistent with the Show response!)

In case of a change action, only the specified component instance and its unchanged parent(s) are published. No unchanged (or changed) child components are published.

In case of a create action, any subcomponents of the specified component instance are also published having action type 'create'.

Limitation: you cannot publish changes on multiple component instances together (for example a change of two order lines, or a create on order line 1 + delete on order line 2).

Note: The ShowAndPublishStandardEvent() cannot be used for specific events (such as ‘approve’). The ShowAndPublishEvent() method cannot be used for standard events (‘create’, ‘change’ or ‘delete’).

Regarding filtering, see Filtering Notes on page 109

Filtering Notes

At runtime, event messages are filtered based on the filter as specified in the SubscribeEvent request. However, in case of standard events, filtering can be a problem for two reasons:

- In case of change or delete events, the event being sent does not contain the complete object including its subcomponents. For delete events, only the identifying attributes from the top-level component are included. For change events, only changed (or created or deleted) components instances and their parent(s) if any are included.
- In case of delete events, the corresponding object or component instance is unavailable, so filtering cannot be done unless only identifying attributes are involved.

For example, when filtering on OrderLine.item = “X”, we cannot apply the filter in case of a delete or change on the Order (header).

For PublishEvent and ShowAndPublishStandardEvent the filtering is done as follows:

- For eventAction ‘delete’, no filter is applied.
- For eventAction ‘change’:
  - For PublishEvent no filter is applied.
  - For ShowAndPublishStandardEvent, no filter is applied if actionType ‘delete’ is used on any component instance. If actionType ‘delete’ is not used, filtering is done.
- For eventAction ‘create’, the filter is always applied.

If the filter is not applied, the event is always published if a subscription exists. In those cases the event consumer may receive irrelevant events.

OnEvent

OnEvent Method

For the OnEvent method, a default implementation is available for:

- standard event actions (create, change, delete);
- specific event actions, in which case the request is forwarded to the corresponding method (unless a different implementation is coded in the OnEvent on execute hook).

For example, a ‘change’ event is by default processed by invoking the ‘Change’ method, and an ‘approve’ event is by default processed by invoking the ‘Approve’ method.

No default implementation is available for ‘list’ events currently. Note that such events cannot be handled by the List method.

Note that the default implementation for handling of create events may be a problem in case of generated data. For example, if an order number is automatically generated in the Create method, but a create event is received from another application which already contains an order number, the order number

from the event is unused. This means the corresponding orders in the applications will have a different number. If you need to avoid this, handle the create event in the on execute hook for the OnEvent method.

Using Action Types in OnEvent

Standard ‘Change’ events and specific events may contain ‘actionType’ attributes to indicate the action (such as ‘create’, ‘change’ or ‘delete’) for a component instance.

The OnEvent method doesn’t check any incoming actionType. In the OnEvent method (which always is a batch method), the request is forwarded to one or more other methods, either by the on execute hook or by a standard mapping from eventAction to method name.

In the underlying method, the action types must be checked. If it is a standard Create or Delete method, action types are ignored. If it is a standard Change method, action types are handled as usual for the Change method. If it is a specific method, action types are handled as just like any other qualifier attribute. Just like any other method arguments, a qualifier attribute will be regarded as input and/or output for the method hooks (before execute, on execute or after execute) if it is specified as input and/or output for the method implementation.

Chapter 6: Modeling BIIs for Complex Cases

When creating a BII, the components, attributes and methods from a BID must be translated to the data and logic in LN and vice versa. This may be a complex job, especially when the BID structure differs significantly from the LN data model or when business logic must be used that is not included in an LN DAL. This chapter describes a number of methods that can be used in such cases.

Screen Scraping using the Application Function Server

'Screen scraping' can be used for implementing the Create, Change and Delete methods.

A Form Importer is available in the LN Studio to import the LN sessions as forms into the LN Studio. When the form is available, it can be used to implement a BID component. This approach is called 'screen scraping', because the business object data is not manipulated through the Data Access Layer, but through the session.

In LN, screen scraping is implemented through the function server, which offers a programming API on top of the LN user interface sessions. This means that for tables of type 'form', the LN Implementation Generator does not generate the normal (database-based) runtime, but a variant, which makes use of function server to 'enter' data, save it, etc.

If a business interface offers other methods than Create, Change, Delete, or if one or two of these methods must not be implemented through function server, the user must make use of an alternative business interface implementation (refer to Alternative Business Interface Implementations on page 118).

Using the Form importer

The Form Importer can be used to import the definition of an LN session into the LN Studio. For details on the Form Importer and its usage, refer to the LN Studio online help.

Some specifics for LN:

- The session code is used as the form's identifier. In case of synchronized sessions (using a multi-occurrence and a single-occurrence session), the Table.name will contain both session codes, separated by an underscore. For example: 'ppmmm0501m000_ppmmm0101s000'. This is also

the case if a single session is used for both the multi- and the single-occurrence screen. For example: ‘ppmmm0101m000_ppmmm0101m000’.

- Just like a database table, a form must have a key (primary index) defined. This is needed to set the corresponding fields for Read, Change and Delete. In case of synchronized sessions, the index will be taken from the first session (which is the multi-occurrence session).
- For a multi-occurrence session the session’s view fields must be known. These are defined in a special index named ‘view’. In case of synchronized sessions, the view will be taken from the first session (which is the multi-occurrence session).
- Display-only fields are irrelevant in the form definition, because a screen scraping implementation can only be used for the Create, Change and Delete methods.
- Regarding interaction points, the form importer for LN does not make use of interaction points. At runtime, questions in LN will get their default answer.

Creating the BII

The LN Studio online help describes how to model using tables of type ‘form’. Additionally, the following limitations must be taken into account for LN:

- Only the Create/Change/Delete methods are supported for screen scraping BIIs. So no List/Show and no other standard or specific methods are supported. A constraint check is done in the LN Implementation Generator to enforce this.
- The BOR Importer doesn't deliver form definitions or screen scraping settings. So the implementation of existing BOR-based screen scraping implementations cannot be migrated in one step to the LN Studio. Instead, the BID can be imported from the BOR and the Form Importer must be used to retrieve the session information from LN. Then a new BII can be modeled in the in the LN Studio.
- Unless specified otherwise in this chapter, the constraints for BOR-based screen scraping implementations also hold for LN Studio screen scraping implementations.
- Screen scraping can only be used for three types of LN sessions/forms: stand-alone sessions type 1 (single-occurrence), stand-alone sessions type 2 (multi-occurrence editable grid) and stand-alone sessions type 3 (same as type 2, but having view fields). Processing sessions (type 4) cannot be used. Also so-called ‘multi main table sessions’ cannot be used.
- Using text fields is unsupported. Also language-dependent text handling is unsupported for screen scraping implementations.
- So-called ‘segmented fields’ as used in some LN sessions cannot be modeled in the LN Studio.
- Using sessions in ‘multi main table’ mode is unsupported.
- At runtime, the processingScope of a request is limited to ‘business_entity’ and ‘business_entity_component’.

Note:

- Using screen scraping will significantly slow down performance.
- Hooks (including method hooks) can be used for screen scraping implementations. You can also use a batch implementation for a method in a screen scraping BII.
- Tables of type ‘form’ can be linked per component implementation. This means that it is possible to implement one component through a normal table mapping, while implementing another component through screen scraping. However, for a single component, database tables and forms cannot be combined.

- Interaction points are unused in the business object runtime. The interaction point and its default value (if any) is handled automatically in the session. Consequently, if the interaction point's default value in the LN Studio differs from the default value as defined in LN, the latter will be used.

Associated non-root tables

Introduction

In the BII, multiple tables can be mapped to a single component. A component will always have one 'root table'. Non-root tables must always be related to another (root or non-root) table. The relations are defined in the BII (there need not be any reference defined in the data model). Cycles or self-references are not allowed. In the end, all non-root tables must be connected to the root table, either directly or indirectly.

Note: Referential integrity tables are a different category. A referential integrity table must by definition have a one-to-one relationship with the root table through their primary keys. Referential integrity tables are not discussed in this section.

Multiple ways exist to link a non-root table to a component's root table or to another non-root table. For the impact on the List and Show methods, refer to the former section "Complex Mappings". The Create, Change and Delete methods are discussed in the following section. A number of examples are given.

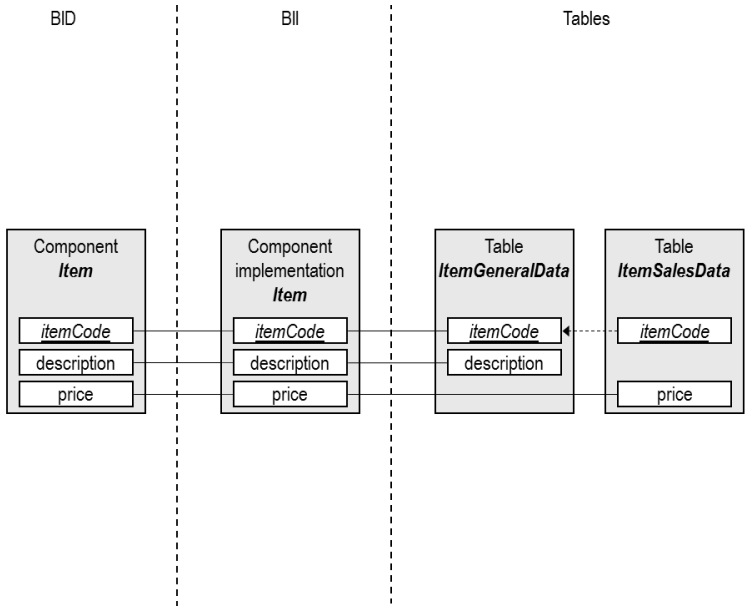
Note that in all cases the implementation will behave as if the data in the non-root table is aggregated in the component. If this is not the desired behavior, do not use a non-root table, but use an association relationship to another business interface instead, or use hooks to implement the complex mappings.

Linking Non-Root Tables

Non-root tables can be linked to other tables in the following ways.

Situation 1: direct mapping through identifiers.

Relationship between root and non-root tables - situation 1:

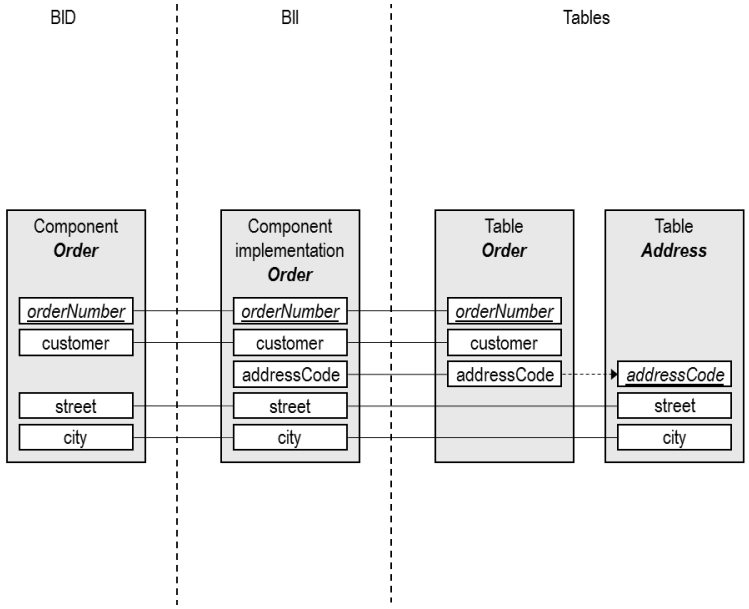


Identifying attributes/columns are shown in italics.

This is the situation that was supported already in Integration Studio 6.1. In this case, the non-root table is really aggregated in the root table. A row in the root table will always exist for a component instance. A row in the non-root table need not exist; it will exist if a price is set or if the table is marked as ‘mandatory’ in the BII. If a component instance is deleted, then the non-root row is deleted first and then the root row is deleted.

Situation 2: association to non-root table using foreign key.

Relationship between root and non-root tables - situation 2:



The address is a typical ‘associated non-root table’ case. Compared to the previous situation, the relationship direction has changed: the root table refers to the non-root table.

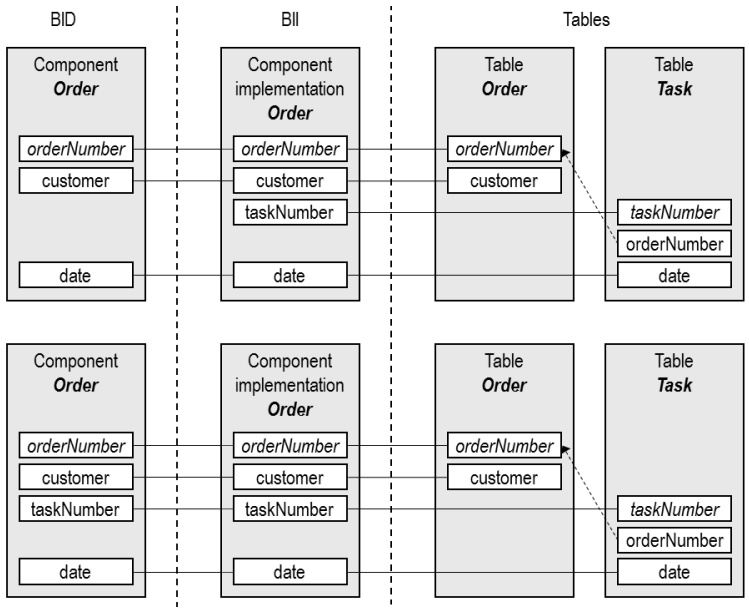
Note: According to this model, a one-to-many relationship exists between the tables. A single Address row can in theory be used in multiple Orders. However, if the Address is modeled as a non-root table, it will be regarded as an aggregated object. That is: for each order that requires an address, a new address row is generated, and that address is changed or deleted if the order is changed or deleted. Addresses won’t be reused.

The BII must make sure that addresses are not reused. If the data can also be changed inside LN, the application must also make sure that addresses are not reused. Otherwise unpredictable results will occur. For example, if an address will be linked to multiple orders, if the Change method is used to change the address for order 1, the address for order 2 will accidentally be changed as well.

If you don’t want the Address to be created, changed and deleted automatically with the Order, do not model the Address as a non-root table. Instead, model it as an associated business object, and/or handle the address-related attributes in hooks.

Situation 3: association to root table using foreign key.

Relationship between root and non-root tables - situation 3:

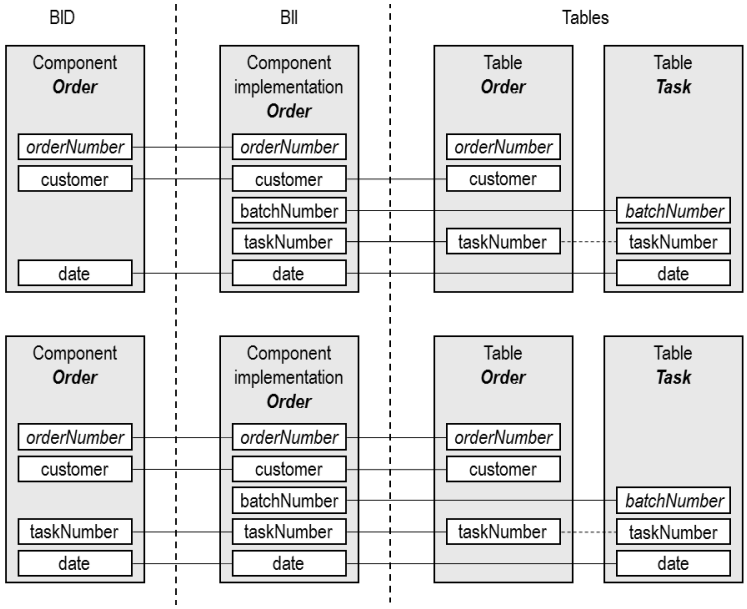


Compared to the previous situation, the relationship direction has changed again: the non-root table refers to the root table.

As shown in the picture, this situation has two variants: the non-root table identifier is either hidden or used as an attribute in the BID. In the first case, the value must be generated. In either case, the implementation (including hooks) must make sure that a one-to-one aggregation relationship is maintained: each order must be in only one task.

Situation 4: association through non-key columns

Relationship between root and non-root tables - situation 4:



The root and non-root table are joined using the taskNumber. In this case the relation can have any direction: refer from root to non-root table or from non-root to root table. For that reason no arrow is shown. Note however that the direction must be defined in the BII!

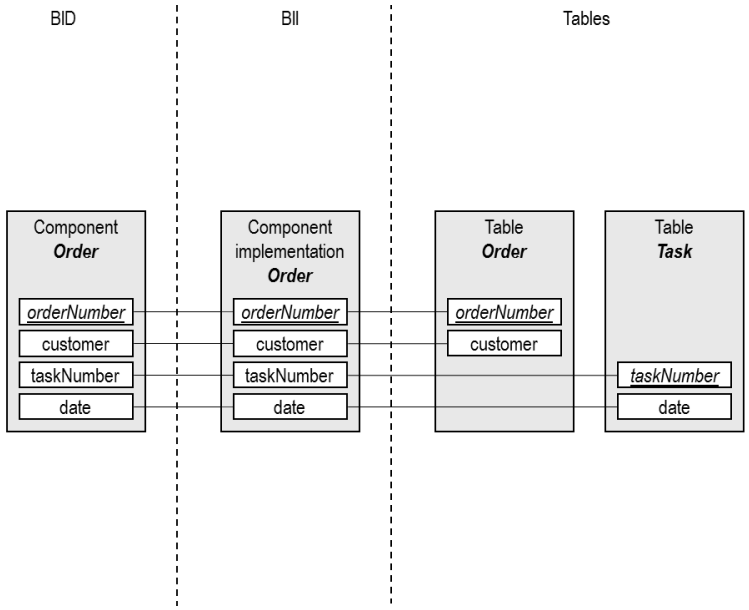
The batchNumber requires a non-related attribute implementation in the BII. Its value must be generated by the implementation when inserting a row on the Task table (during Create or Change).

Note: The value of the attribute(s) that define(s) the relationship must be unique. So for each non-root table row exactly one row exists in the root table, and for each root table row zero or one row exists in the non-root table.

Aside

The following picture shows a situation without a relation between root and non-root table. This situation is not possible, because we will never be able to determine which non-root row belongs to the root row.

Relationship between root and non-root tables - prohibited scenario:



This situation is unsupported, because although no reference needs to be defined in the data model, a join relationship must always be defined in the BII. If multiple tables are linked to a component without such a relationship being defined, then the implementation will not work for either List/Show or Change/Delete methods. For example, a List/Show query will result in the so-called Cartesian product, containing duplicate order numbers, because every task row is joined to every order row. And when changing the date attribute, we won't be able to find the row in the Task table if no join is specified.

For example, when first creating an Order having orderNumber 1 and taskNumber 100, and then creating an Order having orderNumber 2 and taskNumber 200, the database will contain:

Order	Task
1	100
2	200

But since the relationship is undefined, a List will result in four instances (Cartesian product). We don't know whether task 100 or task 200 is related to order 1. And when changing the date attribute for order 1, we don't know what task row to change.

Constraints

In addition to the constraints for the situations as mentioned above, the following general limitations must be taken into account:

- It is not allowed to update an attribute that is mapped to an identifying column on the root table of the component.
- The identifying attributes of all non-root tables have to occur as attribute implementations in the component implementation (either linked to attributes or determined via calculations). Otherwise it is impossible to create, change or delete an instance.

- It is allowed to update an attribute that is mapped to an identifying column on a non-root table, but only if the application allows a row to be deleted and a new row to be created in that case.

Alternative Business Interface Implementations

Overview

A developer can define alternative BIIs for a single BID. This is especially relevant for screen scraping implementations. A common situation is that the List and Show are mapping to database tables, while the Create, Change and Delete methods map to forms (more specifically, tables of type ‘form’).

Alternative combinations are possible. For example, the Delete may go through the standard logic on the table as well. Or it may be implemented using an on execute hook, in which case it will fit in either of both BIIs.

Note that alternative implementations can also be used for other situations than implementations having tables of type ‘form’.

Constraint

An alternative method can only be defined for a complete method. So you cannot implement the Change for the Order component in BII X and for the OrderLine component use an alternative BII Y.

Model

The way to model this in the Integration LN Studio is as follows:

- Use a single BID. Create a public BII for that BID. In fact, the ‘main’ BII for a BID does not need to be public; it can also be protected. If the main BII is a protected implementation, then the BII name must match the BID name. Anyhow, there will always be only one main BII for a BID, having zero to many alternative BIIs. Create an additional protected (or private) BII for the same BID. Whether a BII is public is modeled using the ‘visibility’ property on the BII.
- For those methods that need a different component or attribute mapping, use the protected BII. Implement the methods in that BII.
- In the public BII, create a method implementation for each method. Part of the methods can be implemented directly in the same BII. For example by creating on execute hook for specific methods or by using a default implementation for standard methods. For the methods that are implemented in the protected BII, the public BII will have a method implementation that simply refers to the protected BII. This is done using the alternativeBusinessInterfaceImplementation property on the MethodImplementation.

The public BII must contain method implementations for each method that is supported; the protected BII only needs to implement the methods that are needed there. However, it is not forbidden to create additional protected methods in that BII.

For example:

BID Methods	Public BII Method Implementation	Protected BII Method Implementation
Create	Create -> referring to alternative implementation in protected BII	Create
Change	Create -> referring to alternative implementation in protected BII	Change
Delete	Create -> referring to alternative implementation in protected BII	Delete
List	List, standard implementation	(not implemented)
Show	Show, standard implementation	(not implemented)
DoSomething	DoSomething, on execute hook	(not implemented)

Basically no restriction exist as to which methods are implemented in which BII (except for the constraints as listed below). You can also choose to implement the List and Show in the protected BII and refer to those implementations from the public BII, and implement the other methods in the public BII.

Additionally, you can create multiple alternative BIIs for a single BID.

Constraints

The LN Studio user interface validates the following:

- BIIs being used as alternative implementations are protected (or private). In other words, the can only be (a maximum of) one public BII for a BID. This restriction is needed because a public implementation will use the BID name for the business object name. If multiple public implementations are made, their business object names will clash.
- Method implementations that refer to an implementation in an alternative BII are only implemented for the top-level component. You can say the method has processing order ‘batch’ in the main BII. This is necessary to avoid that the orchestration when executing a method becomes too complex. The public layer will hardly be able to manage the method as a whole if some components are implemented in the same BII and other components in an alternative BII. An alternative method can only be defined for a complete method. You cannot implement the Change for the Order component in BII X and for the OrderLine component use an alternative BII Y.
- For one method there must not be a loop in the implementations. BIIs must not refer to each other (directly or indirectly) for the same method. For the LN implementation generator it doesn’t matter, but at runtime the method will not respond.
- Alternative BIIs are based on the same BID as the public BII. You cannot use a protected BII from another BID as an alternative BII. (Or at least, if a BII from another BID must be used then the interface in the BIDs must be the same for that method.)

Additionally, the following constraints must be taken into account for LN:

- If the Change method exists in a BII then that BII must have Create and Delete as well, at least, if the BID has more than one component. Additionally, if the Change method implementation does not have an alternative BII then the Create and Delete method implementations must also not have an alternative BII.
- An alternative BII cannot be public. Only the main BII can be public. For a public BII the BID name will be used as the business object name in LN; for protected/private BIIs the BII name will be used.

LN Implementation Generator

When generating the runtime for a BII having one or more alternative implementations, the following happens.

Each implementation is generated separately. So the main BII can be generated independent of the alternative BII and vice versa. Note however that the public BII will not work properly until the used alternative BIIs are also generated.

Consequently, you will get a business object in LN for each BII: a lookup entry is created per BII (a single public one and one or more protected ones). Each implementation has its own implementation identifier. A separate set of libraries (public and protected) is generated for each BII. This is needed because the implementation will differ, so the setters, getters, etc. will differ.

Using Multiple Implementations

Introduction

It is possible to create multiple business interface implementations for a single business interface definition in one LN environment, within the same version. For example:

A BID ‘X’ exists. Three BIIs are defined for this BID: a public ‘X’, a protected ‘Y’ and a protected ‘Z’. The X implementation is nothing but an interface that forwards requests to Y and Z. Depending on the value of an attribute X.type, either the Y or the Z implementation is used.

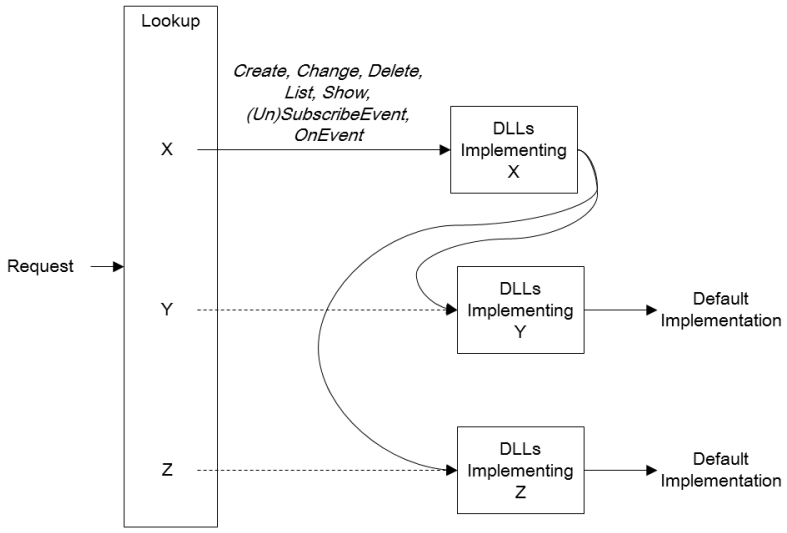
To support this, the following is done:

In LN a single name exists for each business object. Usually this is the BID name. The business object name is used in the lookup and it is the top-level node in the data area. However, if a protected business object is generated, the BII name is used as the business object name.

Note: The user must make sure that in that case the BII name is unique; it must not clash with an existing BID name.

How to Model

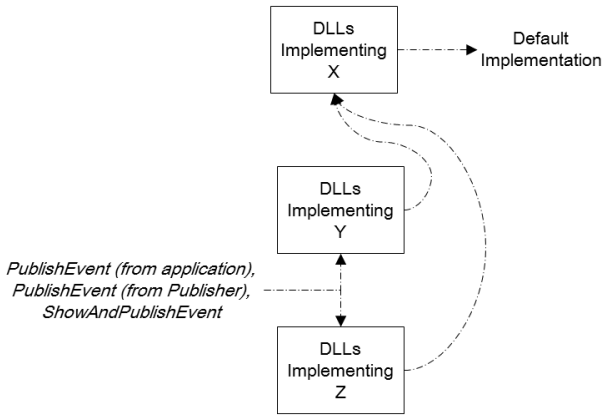
The following figure explains how most methods are handled.



Y and Z do have lookup entries, but they will not be invoked from the outside, because their names do not match existing BID names. Requests for X are forwarded to Y or Z (or both, depending on the method).

Y uses the BII name ‘Y’ as its business object name. It also uses this name for its top-level component. So it doesn’t use the name ‘X’ as defined in the BID. In the same way Z uses ‘Z’ as its name, not ‘X’.

The following figure shows the flow for the PublishEvent() and ShowAndPublishEvent() methods:



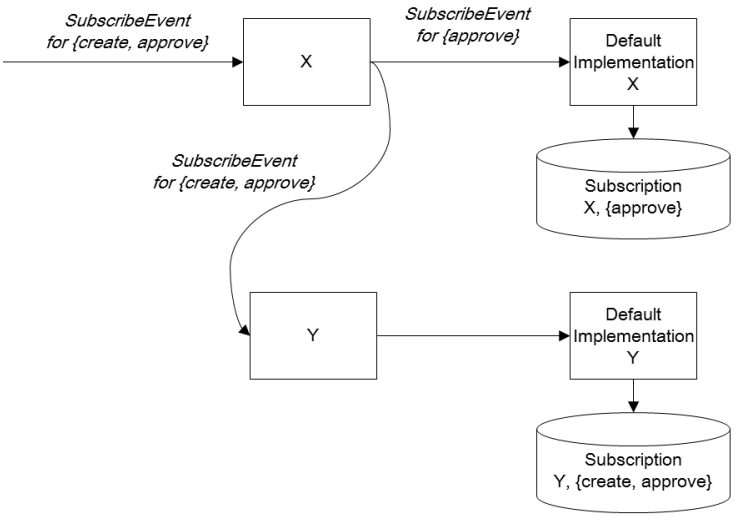
The basic idea for the event handling is as follows:

- Application events are handled in X. This means X also contains the subscription for application events. The application can use the PublishEvent or ShowAndPublishEvent from Y or Z, but in that case Y and Z will do nothing else than forward it to X. The application can also use the PublishEvent from X directly. However, the ShowAndPublishEvent for X usually will not work, because the business interface implementation for X does not have a mapping to tables.
- Standard events (create, change, delete) are handled in Y and Z. This is needed, because the publisher uses the mapping to tables and columns to detect the events.

Example for Event Publishing

The following is an example using a public business object implementation X and a protected business object implementation Y. Note that multiple protected implementations can exist; they will be handled in the same way.

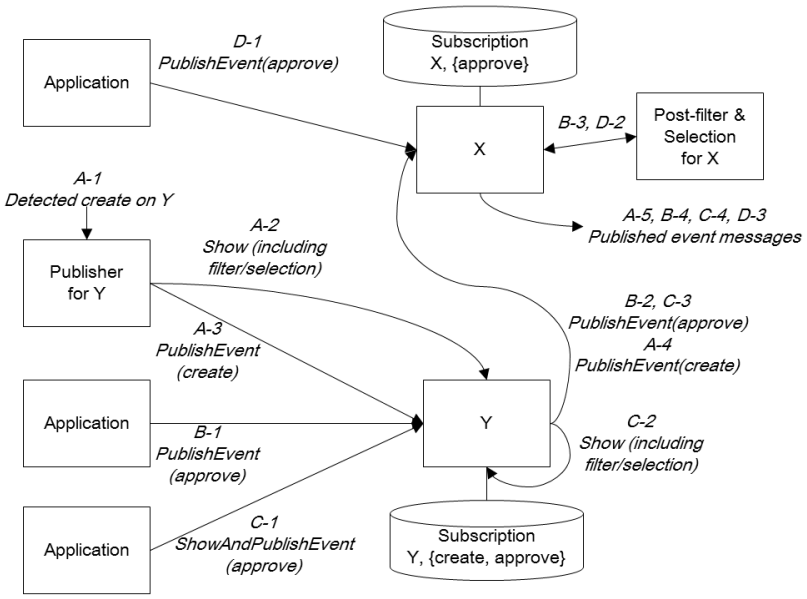
The following picture explains how the subscription is handled.



The following picture explains how the event publishing works based on the subscriptions shown above. The following flows are included in the figure:

- A: Publisher for standard events. For standard events (in this case, create) a publisher is running for Y. This service uses the Show() and PublishEvent() methods from Y.
- B: PublishEvent from the application on Y.
- C: ShowAndPublishEvent from the application on Y.
- D: PublishEvent from the application on X.

Note that the ShowAndPublishEvent() can be done only for Y. It is unavailable for X, because X itself does not have an implementation (mapping to tables).



The solution works because for specific events from ShowAndPublishEvent() and standard events from the publisher, no filtering and selection is done in X. The event is simply published, because the filtering and selection are already done and the destination is already determined. For Y, the postfiltering is unused. Instead the filter and selection from the subscription are included in the Show request as used by the Publisher and the ShowAndPublishEvent implementation

Required Method Hooks

Introduction

The following describes the required logic for the on execute hooks.

Note: Error handling is limited in the example code, to keep the basic logic clear. For the same reason subfunctions are used. Note that these are non-standard functions, which can only be used if they are implemented in a library (or a define inside the hook).

Helper Methods

To be able to easily forward a request to another business object, the following assisting methods are available:

- replaceBdeNameInRequest()
- replaceBdeNameInResponse()
- removeStandardEventsFromRequest()
- getRequest()

For details, see Assisting Functions on page 78.

SubscribeEvent

The on execute hook for the SubscribeEvent of BII X contains the following code:

```
#pragma used dll oppmmmbl002sb00 | business object Y
#pragma used dll oppmmmbl003sb00 | business object Z

#define CHECK_RET(retval, ...)
^  if retval <> 0 then
^   dal.set.error.message(…)
^   return(DALHOOKERROR)
^  endif

long requestY  | request xml for Y
long requestZ  | request xml for Z
long responseY | response xml from Y
long responseZ | response xml from Z
long resultY   | result xml from Y
long resultZ   | result xml from Z
boolean anyEventActionLeft | whether request contains any non-standard events
long retl    | return value to be checked
long duml   | dummy variable

| forward request to Y
retl = replaceBdeNameInRequest(i.request, "Y", requestY)
CHECK_RET(retl)
retl = ppmmm.bl002sb00.SubscribeEvent(requestY, responseY, resultY)
duml = xmlDelete(requestY)
if retl = 0 then
 duml = xmlDelete(responseY)
else
 o.result = resultY
 o.response = 0
 dal.set.error.message(…)
 |* Failed to execute SubscribeEvent for Y which implements X
 return(DALHOOKERROR)
endif

| forward request to Z
retl = replaceBdeNameInRequest(i.request, "Z", requestZ)
CHECK_RET(retl)
retl = ppmmm.bl003sb00.SubscribeEvent(requestZ, responseZ, resultZ)
duml = xmlDelete(requestZ)
if retl = 0 then
 duml = xmlDelete(responseZ)
else
 | limitation: the request for Y already succeeded; we must merge resultZ into
 | the response information area instead?
 o.result = resultZ
 o.response = 0
 dal.set.error.message(…)
 |* Failed to execute SubscribeEvent for Z which implements X
 return(DALHOOKERROR)
endif

| determine request for default handling of X
```

```
retl = removeStandardEventsFromRequest(i.request, anyEventActionLeft)
| note: strictly speaking we are not allowed to change input parameter i.request,
| but we cannot avoid this here
CHECK_RET(retl)
if anyEventActionLeft then
 | note: must not do this if no event actions left, because then we will get a
 | subscription for all standard event actions on X, which is incorrect!
 io.default = true   | to make sure a subscription is created for X for specific
 events
else
 | we are ready
 o.response = xmlNewNode("SubscribeEventResponse")
endif

return(0) | OK
```

Y and Z will get the default implementation for SubscribeEvent().

UnsubscribeEvent

The implementation for the UnsubscribeEvent() is comparable to the implementation for SubscribeEvent(), but a bit simpler. The on execute hook for the UnsubscribeEvent of X contains the following code:

```
#pragma used dll oppmmmbl002sb00 | business object Y
#pragma used dll oppmmmbl003sb00 | business object Z

long retl    | return value to be checked
long duml   | dummy variable

retl = ppmmm.bl002sb00.SubscribeEvent(i.request, responseY, resultY)
retl = ppmmm.bl003sb00.SubscribeEvent(i.request, responseZ, resultZ)
duml = xmlDelete(responseY)
duml = xmlDelete(responseZ)
if resultY <> 0 or resultZ <> 0 then
 o.result = combine.results(resultY, resultZ)  | to be defined; unavailable in
standard
 duml = xmlDelete(resultY)
 duml = xmlDelete(resultZ)
 dal.set.error.message(…)
 return(DALHOOKERROR)
else
 o.result = 0
 io.default = true   | to make sure the subscription is removed for X!
 | Limitation: if no subscription was created for X (because
 | the subscription didn't contain any specific events), a result
 | will be returned even through the method succeeded
 return(0) | OK
endif
```

Y and Z will get the default implementation for UnubscribeEvent().

PublishEvent

X will get the default implementation for PublishEvent().

The on execute hook for the PublishEvent in Y contains the following code:

```
#define CHECK_RET(retval, ...)
^  if retval <> 0 then
^   dal.set.error.message(…)
^   return(DALHOOKERROR)
^  endif

long request  | PublishEvent request for business object X
long retl    | return value to be checked
long duml    | dummy variable (PublishEvent does not return an error value)

retl = replaceBdeNameInRequest(i.request, "X", request)
CHECK_RET(retl, ...)
retl = exec_dll_function("oppmmmbl001sb00", "ppmmm.bl001sb00.PublishEvent",
   duml, request, o.response, o.result)
| We must dynamically invoke this method, because Y is generated
| from the Business Studio before generating X, so a direct invocation
| of X from Y will not compile
| Note: PublishEvent never returns an error value (instead it publishes an error
 message if needed)
| so we only need to check whether the load and exec was successful
CHECK_RET(retl, ...)
duml = xmlDelete(request)
return(0) | OK
```

The implementation for Z is comparable to Y

OnEvent

The OnEvent implementation for X is as follows:

```
#pragma used dll oppmmmbl002sb00 | business object Y
#pragma used dll oppmmmbl003sb00 | business object Z

#define CHECK_RET(retval, ...)
^  if retval <> 0 then
^   dal.set.error.message(…)
^   return(DALHOOKERROR)
^  endif

| implementation is incomplete; see note below

long request  | OnEvent request for business object Y or Z
long retl    | return value to be checked
long duml    | dummy variable (PublishEvent does not return an error value)

type = getTypeFromRequest(i.request)
on case type
case "type1":
 retl = replaceBdeNameInRequest(i.request, "Y", request)
 CHECK_RET(retl, ...)
 retl = ppmmm.bl002sb00.OnEvent(request, o.response, o.result)
```

```
 duml = xmlDelete(request)
 CHECK_RET(retl, ...)
 return(0) | OK
 | no break because of return
case “type2”:
 retl = replaceBdeNameInRequest(i.request, “Z”, request)
 CHECK_RET(retl, ...)
 retl = ppmmm.bl003sb00.OnEvent(request, o.response, o.result)
 duml = xmlDelete(request)
 CHECK_RET(retl, ...)
 return(0) | OK
 | no break because of return
default:
 o.response = 0
 o.result = 0
 dal.set.error.message(…)
 return(DALHOOKERROR)
endcase
```

Note: The OnEvent has the same issue as the Change method, in case the type is changed or unavailable. See the following section Other Methods.

The OnEvent implementation for Y and Z can be the default one.

Other Methods

For Create, Change, Delete and comparable specific methods, the implementation for X is comparable to OnEvent(). But note that those are not processing scope ‘batch’ methods, so the request is not an input parameter for the on execute hook, but it must be retrieved using getRequest().

Note:

- Issue for Change: if the type is changed, the Change request is sent to the wrong implementation (i.e. the implementation that does not yet contain the object instance).

  Solution: try to change Y, if that fails, Show Z, merge Show response and Change request and send as a create to Y.

- Issue for Change: the type may not be included in the request (will not occur in practice if the type is in the top-level component, because all attributes must always be available for the top-level).

  Issue for Delete: we never know the type from the request.

  Solution: try both and at least one must fail.

The ‘batch’ processing scope can also be used for the List and Show methods. For List, Show and specific comparable methods, both Y and Z must be invoked. For List, both responses must be merged, but for Show, only one business object will return a response; that response can be used. Additionally, for both methods a replaceBdeNameResponse(response, “X”, o.response) must be done.

Chapter 7: Import from BOR

Introduction

The Business Object Repository (BOR) in LN contains business object definitions. Until Enterprise Server 8.2, the BOR was used to develop business objects for LN. From 8.3 onwards, the Integration Studio and LN Studio are used instead.

The Business Object Repository is available in 8.3 and higher releases to allow maintenance on business objects developed in previous versions or feature packs. It is not advised to use the Business Object Repository for developing new business objects.

Business object definitions can partially be migrated to the LN Studio through the Import from BOR feature.

The Import from BOR feature is not designed for generating proxies or WSDL. It can however be used for that purpose, but in that case note that:

- Only for business objects that were created in the BOR can be imported.
- The Import from BOR imports both the BII and the BID, while only the BID is needed for generating proxies or WSDL.

Preparations

The Import from BOR can only be used if you have got development authorizations. In other words, you can only import business objects if you are also able to view the corresponding definitions in the **Business Objects (ttadv7500m000)** session.

Before using the Import from BOR, specify the connection to the runtime repository in:

Window > Preferences > Infor LN JCA Connectivity

On the Runtime Repository page, specify the details of the LN environment that contains the business objects to be imported. In the **Company** field, use company 000.

Finally, an interface project must be available or created to import the business object into.

Importing from BOR

1. Right-click the project that must contain the imported definitions and select **Import…**.
2. Select **`BOR from Runtime Repository`** and click **Next**.

   If the connection to the LN environment is set up correctly (see Preparations on page 128), you will get a list of all business objects existing in the BOR. The same business object name may occur multiple times, if multiple versions of the business object exist.
3. Select the business object you want to import. You can select multiple business objects, but do not select different versions of the same business object to avoid overwriting.
4. Click **Finish** to start the import.

After the Import

Note: The imported definition is just a starting point for remodeling the BID and BII.

The Import from BOR does not import all aspects for a business object as modeled in the BOR. This is caused by the differences between the model as used in the BOR and the model as used in the LN Studio.

Basically the following aspects are imported:

- The complete BID including the used data types.
- The tables used in the business object as defined in the BOR.
- The BII, including the attribute implementations for the attributes and the attribute implementations that map directly to table columns. For calculated attribute implementations, the mapping is set to ‘not applicable’.

Some examples of limitations:

- Complex mappings (using calculations) are not imported. Also ‘used attributes’ are not defined. Attribute implementations for calculated attributes will get type ‘notApplicable’.
- A number of constructions that were allowed in the BOR cannot be used in the LN Studio. For example, business objects can be modeled without a component, the name of the main component can differ from the business object name, and the structure of components and attributes can be different per method.
- Usually a BID is independent of the implementation. So for example, the data types used in the BID do not have a native data type specified. The data types used in the BII must have a native data type. When using the Import from BOR
- The imported definition does not always follow the (naming) conventions that are used when defining a business interface in the LN Studio from scratch.

Consequently, after importing you must adapt the BID as needed, complete the BII, and solve problems (warnings or errors) as reported.

Compatibility at Runtime

To some extent, the Business Object Layer (BOL), which is the business object runtime based on the BOR is compatible with the business object runtime as generated from the LN Studio.

However, migration to the LN Studio cannot always be done without any impact on the interface at runtime. For example:

- As mentioned in After the Import on page 129, some interfaces allowed in the BOR cannot be defined using the LN Studio.
- Event publishing functions have changed. In Enterprise Server 8.2, PublishChanges, UnpublishChanges and PublishList were used. These functions can still be used. But if the business object is migrated from the BOR to the LN Studio, the new functions (SubscribeEvent, UnsubscribeEvent, SubscribeList, and PublishEvent) must be used.

Chapter 8: BDE web services

For each generated BDE, you can use a SOAP-based web service. When you generate the BDE implementation, also a WSDL is generated. This WSDL is stored on the LN system.

You can deploy the generated BDE on the **Deployed Web Services** administration page in the Infor Enterprise Server Connector for Web Services. Deploying a BDE is not required in an MT environment.

For more information about the connector, see the *Infor Enterprise Server Connector for Web Services - Administration and User Guide*.

In an on-premises environment, you can download the BDE WSDL from the **Web Services Status** administration page in theConnector for Web Services. In an MT environment, you can download the WSDL from the LN system through the **Business Objects (ttadv7500m000)** session.

Based on the WSDL file, you can create your own client application to run BDE methods on an LN system.

When using Infor ION API on premises, you must create an entry in the available APIs for LN in the ION API administration pages.

Chapter 9: Implementing OAGIS BODs for LN

Caution: With Enterprise Server 8.4.2, BOD-related functionality is not generally available. Do not create BODs or BOD-based integrations without prior written consent from Infor. No support will be given on the use of BODs (unless agreed otherwise).

The LN Studio is used to create business interface implementations for LN. In addition to BDE implementations also BOD implementations can be created.

BOD implementations differ from BDE implementations with respect to the offered methods. BOD implementations are able to receive and publish BODs, while BDEs use a request-response paradigm and offer standard methods such as Create, Change, Delete, List and Show. However, BOD implementations are very similar to BDE implementations with respect to the noun part (the data structure of components and attributes) and with respect to the technical implementation in LN.

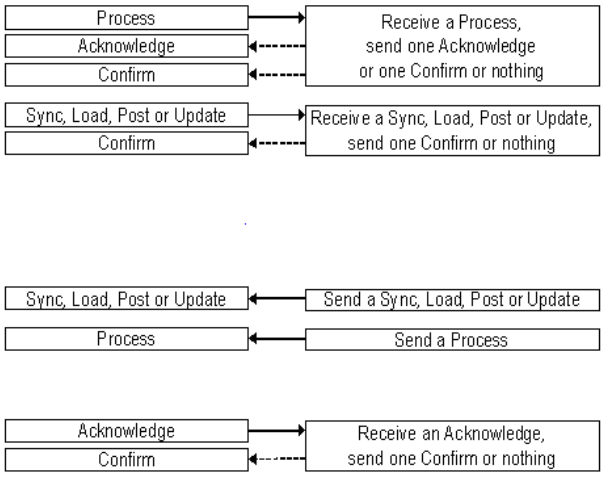
This chapter explains how to create business interface implementations for handling BODs in LN:

- Overview on page 132 provides an overview of the way BODs are handled in LN.
- Modeling and Implementation on page 134 describes how to model the business interface for a BOD using the LN Studio.
- Publishing BODs from the LN Application on page 146 explains how to publish BODs from the LN application.

Overview

Supported Message Flows

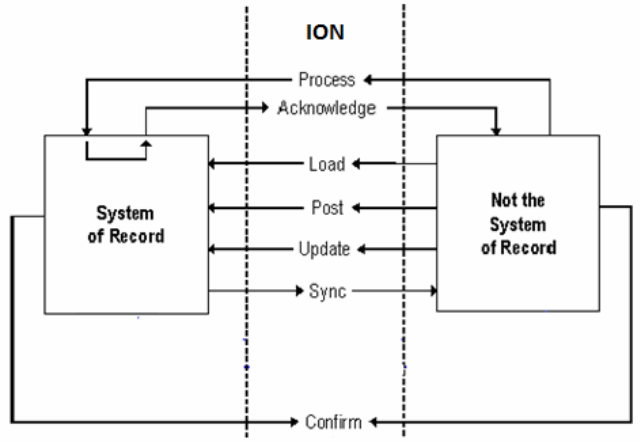
The following message flows are supported in LN:



When an error occurs and no ConfirmBOD is published (and the error is also not reported back to the sending LN application), an error message is published.

Note: The Get/Show cycle is currently unavailable. You cannot send or receive Get BODs or Show BODs in LN.

Each LN company is an application instance in the network. It will have its own connection point to ION. An application instance can be the owner for a specific object or object instance. In that case it is called ‘system of record’ (SOR). The following figure shows the messages that can be sent from or to a SOR. Error messages (ConfirmBODs) are always handled by the CSI (Common Services Infrastructure) application. Note that the CSI can resend rejected messages to one of the applications; this is not displayed in the following picture.



Receiving BODs in LN

When a BOD from ION arrives at the ION LN Adapter instance for an LN company, the incoming BOD is handled as a business object request. The verb instances for the BOD are unchanged, only the format of the ‘envelope’ (control data) is changed to match the business object runtime. The business

object name and method are derived from the BOD noun and verb. The request is then forwarded to the dispatcher. The dispatcher's BDE method interface can also be called directly for non-BOD BDEs. This is done by the OpenWorld Adapter for LN..

The dispatcher finds the specified business object implementation and has it execute the method. The method returns a response (if success) or a result (if error). If applicable, the response or result is translated to a reply BOD (such as Acknowledge or Confirm), and is published to the reply destination as specified by the ION LN Adapter.

The incoming BOD message is only removed from the incoming queue after completely processing the incoming BOD. So it remains in the queue until the ION LN Adapter receives a reply that it can be deleted. That is, until the business object method has been executed and the reply BOD (if any) has been published successfully. So the flow from the ION LN Adapter to LN is synchronous.

When the process for a BOD message has ended, the ION LN Adapter can take the next message from ION, to have it processed in the same way.

Publishing BODs from LN

Events are published from LN through the standard business object event publishing interface. When an event is published for a business object that implements a BOD interface, then the message is formatted as a BOD. Otherwise it is published as a BDE event.

BODs are published to ION.

If LN is in an event producer role, a subscription is needed for each business object that needs to publish. In the BDE paradigm, the SubscribeEvent and UnsubscribeEvent methods are available for this. In the BOD paradigm, 'subscriptions' are created automatically from outside LN.

Regarding error handling, any error that occurs in LN while trying to create or publish a message, is returned to the application. (This is different compared to BDEs, where error messages are sent to the LN Adapter, which will publish or log the error.)

Modeling and Implementation

Overview

Business Object

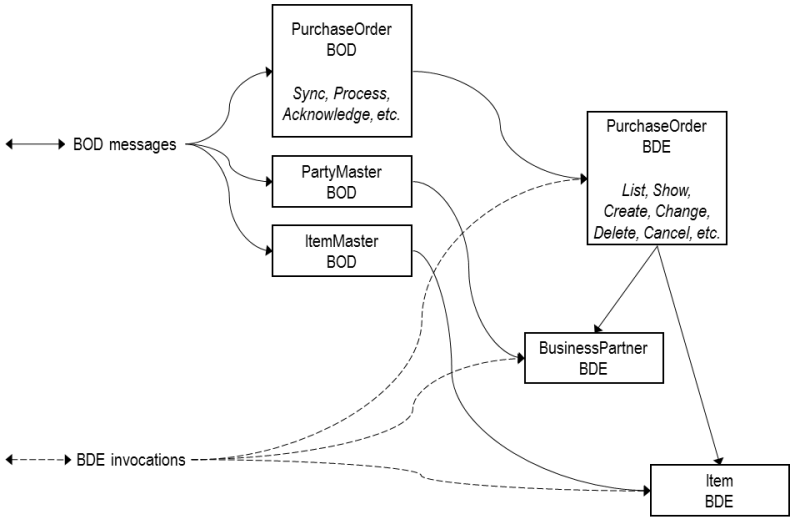
To implement the BODs for an OAGIS noun, a public BII is used. The BID will use the OAGIS noun as its name. The public implementation for this BII in LN will have a name that is equal to the noun with a postfix 'BOD'. A protected implementation for a BID of type BOD can have 'any' name.

The postfix is used to avoid clashes with existing business objects (such as ‘PurchaseOrder’). For BIDs of type BOD, the corresponding business object in LN will always use the BII name as its name, also for public BIIs. Additionally, the LN Implementation Generator will report an error if the BID is of type BOD and the BII is public, but the BII name is unequal to the BID name with a postfix ‘BOD’.

Hint

It is advised to use multiple BIDs and BIIs to implement a BOD noun. In the first place, a BOD noun usually is a complex structure, combining multiple associated objects. For example, an order contains customer or supplier data, address data, item data, etc. Creating multiple business objects for this helps to implement the BOD in a structured way. In the second place, not only associated objects need to be separate business objects, but also the main object can be a separate BDE type of business object. This enables reuse of the BOD noun for BDE-based integrations. And it makes testing easier, because you can test underlying methods such as Show or Create, which otherwise cannot be tested from the LN Studio Test Tool, because in the BOD they will not be a part of the public interface.

An example is shown in the following picture:



Events

For BOD BIDs, events must be modeled in the LN Studio user interface. The developer specifies which messages can be sent or received for a BID. No public methods are modeled.

In the BII model, events are not modeled explicitly. A fixed, predefined relation exists between the events in the BID and the methods in the BII. No ‘event implementation’ entity is introduced in the BII. Consequently, the mapping of events to method implementations is implicit. The LN Studio user interface will report a problem if the method implementations in the BII do not match the modeled events from the BID.

Both sending and receiving events is done through method implementations.

The required implementations are as follows:

Event in BID	Implementation for 'Can Be Published'	Implementation for 'Can Be Received'
Process	PublishEvent and ShowAndPublishProcessBOD	OnProcess
Acknowledge	n/a (handled through OnProcess)	OnAcknowledge
Sync	PublishEvent and ShowAndPublishSyncBOD	OnSync
Load	PublishEvent and ShowAndPublishLoadBOD	OnLoad
Post	PublishEvent and ShowAndPublishPostBOD	OnPost
Update	PublishEvent and ShowAndPublishUpdateBOD	OnUpdate
Get	unsupported	unsupported
Show	unsupported	unsupported
Confirm	n/a (Confirm BODs are sent through the On methods)	n/a (LN cannot receive Confirm BODs)

Overview of Methods

A number of predefined verbs are used for BODs: Sync, Load, Post, Update, Process, and Acknowledge. Depending on the situation, the business object will have to be able to receive such BODs, or send them, or both.

For receiving BODs, a method On<Verb> must be implemented, for example, OnSync, OnProcess, OnAcknowledge. The 'On' is added to make clear that the methods are for receiving BODs, not for sending BODs, to avoid a name clash for the existing standard Show method and to avoid confusion with existing specific methods. For sending BODs, the event publishing methods are used.

The business object offers the following methods (or a subset):

- Methods for receiving requests: OnSync, OnLoad, OnPost, OnUpdate, OnProcess,
- Methods for receiving replies on sent requests: OnAcknowledge (used after sending a Process request).
- Method for sending events or request BODs: PublishEvent. This method must be implemented if a Process, Sync, Load, Post or Update event can be published. The ShowAndPublish<Verb>BOD methods (see following section Publishing BODs from the LN Application) are not modeled explicitly, but will automatically available if the PublishEvent is available.
- For using the ShowAndPublish<Verb>BOD methods, also the Show method must be available.

Each of these methods will be protected. The SubscribeEvent and UnsubscribeEvent methods (as used for BDE event publishing) are not used for BODs.

The business object implementation can make use of (protected) standard methods such as Create, Change and Delete. These can be helpful to easily implement the OnSync, OnLoad, etc. These methods can either be included in the same implementation as protected methods, or they can be implemented in a separate business object.

Methods for Incoming BODs

Incoming ‘Request’ BODs

The following business object methods are needed for receiving request-type BODs in LN.

Incoming BOD	Required Method	Reply BODs
Process	OnProcess	Acknowledge or Confirm or none
Sync	OnSync	Confirm or none
Load	OnLoad	Confirm or none
Post	OnPost	Confirm or none
Update	OnUpdate	Confirm or none

For the OnProcess method the developer must make sure that:

- The method returns a response (so not a result) if an Acknowledge BOD must be sent.
- In the response, a controlling attribute ‘actionCode’ is available to indicate the ActionExpression.actionCode for the acknowledge BOD.
- The method returns a result if and only if a Confirm BOD must be sent.

The actionCode must be set in a method hook for the OnProcess method using:

```
ret.bool = setControlAttribute(“actionCode”, actionCodeValue)
```

Or, in case of a batch implementation for the OnProcess method:

```
ret.bool = setControlAttributeInResponse(o.response, “actionCode”,
  actionCodeValue)
```

If these functions are invoked multiple times, only the last value is used. If it is unused, no acknowledge BOD will be sent.

The setControlAttribute or setControlAttributeInResponse can be used to include any control element (except complex XML structures).

Regarding the actionCode, note the following:

- The actionCode for an Acknowledge BOD will be: ‘Accepted’, ‘Modified’ or ‘Rejected’. The value is used for the BOD as a whole, so it cannot be set per object instance. So the response cannot indicate that on order instance is ‘Modified’ and another one is ‘Accepted’. In that case the actionCode will simply be ‘Modified’.
- If the actionCode is empty, then no Acknowledge BOD is sent. This is achieved by using setControlAttribute("actionCode", "") Reason: if a business object method has output arguments, then the standard business object runtime will always create a response. This means that it will be impossible for the application to send an Acknowledge in some cases, while not sending an

Acknowledge (and not sending a Confirm either) in other cases. Therefore the application is allowed to use an empty actionCode to suppress the Acknowledge message, if needed.

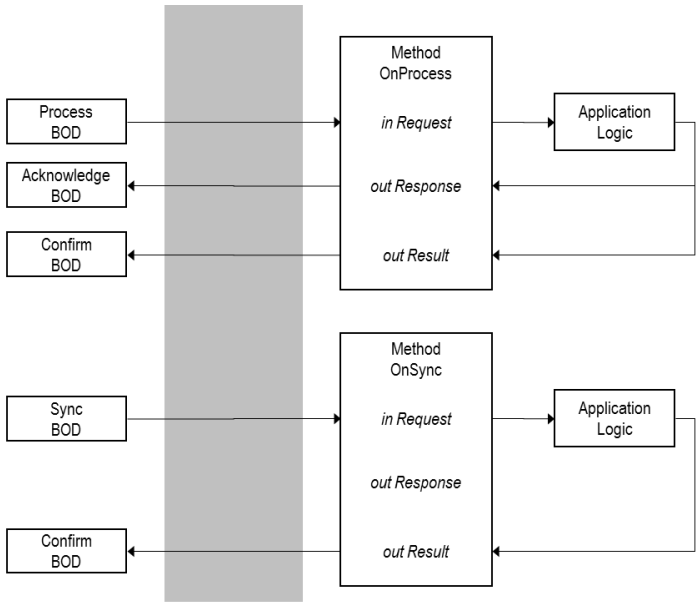
- In case of ‘Accepted’ or ‘Rejected’, the noun portion can be empty. However, this is unsupported in the standard business object runtime. If the noun must be empty, then the OnProcess implementation must make sure no data is included in the response. However, in practice data will be needed, because the receiving party must at least be able to identify the involved noun instance. The reason for that is that at this moment we are unable to link the received Acknowledge to a sent Process otherwise.Anyhow, the LN implementation can set the actionCode to ‘Modified’ if needed, for example if the value for one or more attributes is generated within LN, or if it is desirable to send an Acknowledge anyhow.
- Additionally, in case of ‘Modified’ the noun must only include changed elements. However, this is not done automatically by the standard logic. By default, all output attributes are included. So if needed it must be programmed in the on execute hook for the OnProcess method (or an underlying protected method).
- Regarding the distinction between an Acknowledge having actionCode ‘Rejected’ and a Confirm BOD: if a Process request resulted in a state change in LN, use Acknowledge, otherwise use Confirm. When receiving an Acknowledge having status 'rejected' on the requesting side, a user can ask 'Why has purchase order 1234 been rejected?' And the LN user can check the system, find the purchase order and explain the situation. In case of a Confirm, the database transaction was aborted. The instance is not stored at the system of record. Note that this matches the distinction between ‘response’ and ‘result’ in BDE terms: in case of a result, the transaction is aborted.

For other methods the developer must make sure that:

- The method never returns any output in the response (except warnings in the information area).
- The method contains one or more messages in the response information area if warnings must be logged, but no Confirm BOD must be sent.
- The method returns a result if and only if a confirm BOD must be sent.
- In the ‘On’ methods, the ActionExpression.actionCode from the incoming BOD can be retrieved in any BII hook using:

```
ret.bool = getControlAttribute(“actionCode”, actionCodeValue)
```

To summarize, the following picture shows the flow for two typical ‘from bus’ scenarios: the Process/Acknowledge cycle and the Sync handling.



Incoming ‘Reply’ BODs

If LN published a Process BOD, then it may asynchronously receive a BOD in reply. The following table lists the method that is needed for receiving replies on requests that were sent from LN.

Incoming BOD	Required Method	Reply BODs
Acknowledge	OnAcknowledge	none

For this method the developer must make sure that:

- The method never returns any output in the response (except warnings in the information area).
- The method contains one or more messages in the response information area if warnings must be logged.
- The method returns a result if and only if a confirm BOD must be sent.

Note: Confirm BODs will not (or at least: must not) arrive at LN. Consequently, no OnConfirm method is needed. An OnAcknowledge method is only needed if LN publishes ‘process’ events for that business object.

Again, the getControlAttribute() function can be used to retrieve the ActionExpression.actionCode, see Incoming ‘Request’ BODs, above.

An Acknowledge that is received is related to a Process request that was sent before. The OnAcknowledge method must be able to determine that relation.

In some cases the (identifying) attributes will be sufficient for the application to handle the incoming Acknowledge. But this will not always be the case, because the application that received the Process and sent the Acknowledge may have generated a new identifier or returned an already existing object. In that case the LN application can determine the relation between Process message and Acknowledge message as follows.

When sending a BOD using ShowAndPublishProcessBOD, you can specify a three-part ID (see following section “Details”). In the OnAcknowledge, if you create a batch implementation you can use one or more of the following lines in the on execute hook:

```
ret.bool = getControlAttributeFromRequest(i.request,
     "originalDocumentID", original.document.id)
ret.bool = getControlAttributeFromRequest(i.request,
     "originalRevisionID", original.revision.id)
ret.bool = getControlAttributeFromRequest(i.request,
     "originalVariationID", original.variation.id)
```

This will retrieve the information from the OriginalApplicationArea of the Acknowledge BOD. So the resulting IDs must match the IDs as set in the ShowAndPublishProcessBOD method.

In case of a top-down implementation, you can use the getControlAttribute function instead of getControlAttributeFromRequest.

Methods for Outgoing BODs

The following table lists the methods that are needed for sending BODs from LN.

Outgoing BOD	Required Methods
Request BODs (Process, Sync, Load, Post, Update)	PublishEvent, Show; when sending Process or Get BODs, the OnAcknowledge is also needed for handling the incoming reply BOD (see section Methods for Incoming BODs).
Reply BODs (Acknowledge, Confirm)	none

To be able to send request BODs, the business interface (BI) must offer the PublishEvent method. In the model and its implementation, the developer must make sure that the method complies with the Business Data Entity Implementation Standard and Event Management Standard.

The Show method is needed to enable the LN application to publish BODs using ShowAndPublish<Verb>BOD methods.

For BODs, publishing is done from the LN application, so the application must be adapted to be able to publish the required events. See Publishing BODs from the LN Application on page 146.

The PublishEvent method can be used to publish the following events (or a subset): Process, Sync, Load, Post, and Update. It is not needed to publish Confirm or Acknowledge events through the business object, because those BODs are created automatically while processing an incoming request BOD.

The table provides an overview for the events that can be sent and how this is done:

Publishing Method	Event Action(s)	Outgoing BOD (Verb)
PublishEvent	Process	Process
	Sync	Sync
	Load	Load
	Post	Post
	Update	Update
ShowAndPublishProcessBOD	n/a	Process
ShowAndPublishSyncBOD	n/a	Sync
ShowAndPublishLoadBOD	n/a	Load
ShowAndPublishPostBOD	n/a	Post
ShowAndPublishUpdateBOD	n/a	Update

When using PublishEvent, both the verb (eventAction) and actionCode are explicitly specified by the sending LN application.

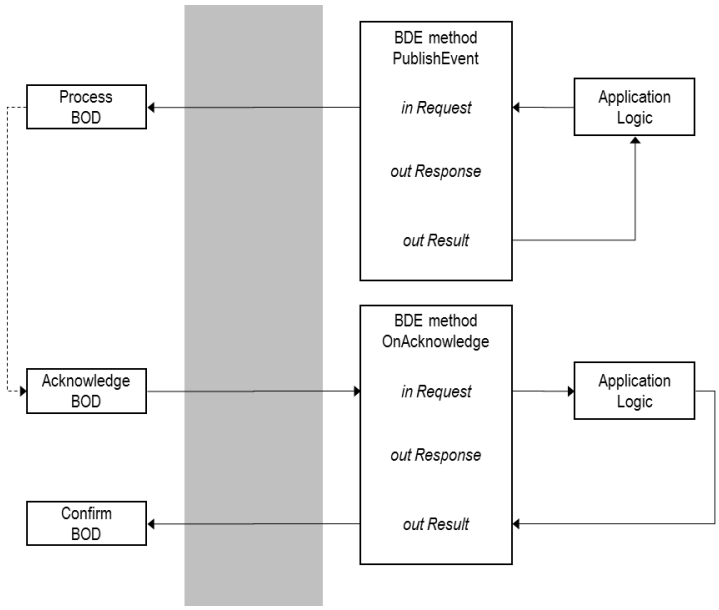
When using ShowAndPublish<Verb>BOD, a separate publishing function exists for each verb.

Note: In case of BDE change events, each business object and component instance will have an actionType attribute to indicate the action at that level. The actionType will be ‘create’, ‘change’, ‘delete’ or ‘unchanged’. These action types are not allowed for BODs according to the OAGIS XSDs, so they are not included. This implies however that part of the semantics will be lost.

For the BOD, the interpretation might be:

- If only identifiers are included for a business object or component then the action type is ‘delete’.
- Otherwise the action type is undefined (it may be create, change or unchanged). When using a protected Change method to process the BOD, the createOrChange action type may be useful (see Action Types on page 35).

To summarize, the following picture shows the flow for a typical ‘to bus’ scenario.



LN sends a Process BOD and after some time (asynchronously) can receive an Acknowledge BOD. Although technically two things happen at the application server (sending a BOD and receiving a BOD), functionally this is a single scenario.

Method Arguments

The following table lists the input and output attributes that are relevant for BOD methods.

Method	Input/Output	Input only	Output only
OnProcess	All public attributes except those listed for 'output only'	none	Public attributes for which the value is always determined inside LN
OnSync	none	All public attributes except attributes for which the value is always determined inside LN	none
OnLoad	none	same as OnSync	none
OnPost	none	same as OnSync	none
OnUpdate	none	same as OnSync	none
OnAcknowledge	none	same as OnSync	none
PublishEvent (protected)	none	All public attributes	none
List (protected)	none	none	All public attributes

Method	Input/Output	Input only	Output only
Show (protected)	Identifiers for top-level component	none	All public attributes
Create (protected)	All public attributes except those listed for 'output only'	none	Public attributes for which the value is always determined inside LN
Change (protected)	All public attributes except those listed for 'output only'	none	Public attributes for which the value is always determined inside LN
Delete (protected)	none	Identifiers for top-level component	none

The method arguments for the 'On' methods do not have to be modeled explicitly in the LN Studio. They are handled automatically.

Helper functions

As mentioned earlier in this chapter, a number of helper functions are available which can be used in hooks. These functions are discussed in Assisting Functions on page 78.

Examples

Sending a BOD

Methods PublishEvent and Show are available as protected methods. No specific implementation is needed.

For code that is required for publishing any event from the LN application, refer to Publishing BODs from the LN Application on page 146.

Receiving a BOD

To be able to receive a Sync BOD, a public OnSync method is modeled. This method is implemented on the top-level component, using processing scope 'batch'. Additionally protected methods Create, Change and Delete are available.

Assuming the implementation identifier is ppmmm123, the on execute hook for the OnSync method contains the following code:

```
 | Important: this is a hypothetical example for an OnSync hook,
 | for educational purposes only.
 | If is not necessarily a realistic, optimal or advised implementation.

 table tppmmm123 | orders

 domain ppmmm.docu document.id | first identifying attribute
 domain ppmmm.revi revision.id | second identifying attribute
 domain ppmmm.docu dum.doc.id | dummy variable to avoid change of record buffer

 string action.code(20) | action to be done, either “add”, “change” or “delete”
or
       | empty/missing
 long ret.method  | return value from business object method
 long ret.exec   | return value from exec_dll_function

 #define EXEC_METHOD_AND_RETURN(method_name)
^  | we must dynamically invoke methods to avoid a dependency loop
^  | between public and protected library.
^  ret.exec = exec_dll_function(“oppmmmbl123sb00”,
^     “ppmmm.bl123sb00.##method_name##”,
^     ret.method, i.request, o.response, o.result)
^  if retl.exec <> 0 then
^   dal.set.error.message(“ppmmm.0003”, “##method_name##”)
^   |* Unable to handle incoming Sync message because method %s is
^   |* missing in OrderBOD
^   return(DALHOOKERROR)
^  else
^   return(ret.method) | either 0 (OK) or DALHOOKERROR
^  endif

 if not getControlAttribute(“actionCode”, action.code) then
  | no problem, action.code is empty
 endif

 if strip$(tolower$(action.code)) = “delete” then
  | invoke Delete method
  EXEC_METHOD_AND_RETURN(Delete)
 else
  | action code is ‘Add’, ‘Change’ or undefined;
  | determine required action based on existence of object instance

  | get identifying values
  if not getAttributeValueFromRequest(i.request,
     “OrderBOD. Header.ID.DocumentID”, document.id)
  then
   dal.set.error.message(“ppmmm.0001”)
   |* Incoming Sync message does not contain DocumentID
   return(DALHOOKERROR)
  endif
  if not getAttributeValueFromRequest(i.request,
     “OrderBOD. Header.ID.RevisionID”, revision.id)
  then
   | no problem; assuming revision id can be empty
  endif

  | check existence of order
  select ppmmm123.docu:dum.doc.id
```

```
 from ppmmm123
 where ppmmm123.docu = :document.id and ppmmm123.revi = :revision.id
 as set with 1 rows
 selectdo
  | order already exists; invoke Change method
  EXEC_METHOD_AND_RETURN(Change)
 selectempty
  | order does not exist; invoke Create method
  EXEC_METHOD_AND_RETURN(Create)
 endselect

 | we must not get here…
 dal.set.error.message("ppmmm.0002", document.id, revision.id)
 |* Error reading order '%s-%s' from the database
 return(DALHOOKERROR)
endif
```

A simplified example for the on execute hook of an OnProcess method (again a 'batch' implementation), to illustrate the setting of the actionCode, which makes sure that an Acknowledge BOD is sent in reply:

```
| Important: this is a hypothetical example for an OnProcess hook,
| for educational purposes only.
| If is not necessarily a realistic, optimal or advised implementation.

long ret.method  | return value from business object method
long ret.exec    | return value from exec_dll_function

| we will dynamically invoke the Create method to avoid a dependency loop
| between public and protected library.
ret.exec = exec_dll_function("oppmmmbl123sb00", "ppmmm.bl123sb00.Create",
   ret.method, i.request, o.response, o.result)
if retl.exec <> 0 then
 dal.set.error.message("ppmmm.0004")
 |* Unable to handle incoming Process message because Create method is missing
 |* in OrderBOD
 return(DALHOOKERROR)
else
 if ret.method <> 0 then
  dal.set.error.message("ppmmm.0005")
  |* Unable to handle incoming Process message because Create method in
  |* OrderBOD returned an error
  return(DALHOOKERROR)
 else
  if not setControlAttributeInResponse(o.response, "actionCode", "Modified") then

   dal.set.error.message("ppmmm.0006")
   |* Cannot set actionCode; unable to send Acknowledge reply for incoming
   |* Process message
   return(DALHOOKERROR)
  else
   return(0) | OK
  endif
 endif
endif
```

Publishing BODs from the LN Application

Overview

Publishing Methods

This section describes how to adapt the LN application to publish BODs. A number of predefined methods is available to publish from LN.

No interface is available publishing for reply BODs (Confirm, Acknowledge), because those are handled automatically. For request BODs, the following event publishing methods can be used:

- PublishEvent;
- ShowAndPublishSyncBOD, ShowAndPublishProcessBOD, ShowAndPublishLoadBOD, ShowAndPublishPostBOD, ShowAndPublishUpdateBOD. In the remainder of this document, these methods can be referred to as ShowAndPublish<Verb>BOD.

For BOD-type business objects, the ShowAndPublish<Verb>BOD methods must be used instead of ShowAndPublishEvent or ShowAndPublishStandardEvent.

Which Verb to Use

Section Methods for Outgoing BODs on page 140 lists the events that can be published for BOD-type business objects.

The LN application must not publish Confirm, Acknowledge, or Show BODs.

Regarding the difference between Sync on the one hand and Load, Post, and Update on the other:

- LN must send a Sync for business object instances for which it is the ‘system of record’. In other words, each instance that was created inside the LN instance.
- LN must send a Process, Load, Post or Update if an instance needs to be created outside the current LN instance, or if an instance that is owned outside the current LN instance needs to be changed.

When implementing LN and other applications at a customer site, ownership may differ per business object (class), but also for object instances of the same class. For example: items are always created in company 100 and sales orders are created 200 or 300 where each company uses another range of order numbers.

Details

ShowAndPublishEvent

For business objects of type ‘BOD’, the ShowAndPublishEvent must not be used. Use the ShowAndPublish<Verb>BOD methods instead.

ShowAndPublish Verb BOD

For business objects of type ‘BOD’ having the PublishEvent method implemented, the following methods are also available:

ShowAndPublishSyncBOD, ShowAndPublishProcessBOD, ShowAndPublishLoadBOD, ShowAndPublishPostBOD, ShowAndPublishUpdateBOD.

However, these functions will only be available for the events that are marked as ‘can be published’ in the BID.

The interface is as follows:

```
long ppmmm.bl999st00.OrderBOD.ShowAndPublish<Verb>BOD(
 domain    ppmmm.orno  i.internalOrderNumber,  | for example
 const     string      i.actionCode,
 const     string      i.documentId,
 const     string      i.revisionId,
 const     string      i.variationId,
 const     string      i.tenantId,
 const     string      i.accountingEntityId,
 const     string      i.locationId)
```

The behavior of the ShowAndPublish<Verb>BOD method is comparable to the ShowAndPublishEvent (see ShowAndPublishEvent on page 103), except that additional details as needed for BODs can be set, the error handling differs, and the ShowAndPublish<Verb>BOD methods are unavailable for subcomponents.

The methods must be called from within a database transaction.

The objects are published as they are at the moment of publishing. So if an object is changed and that event must be published, the publishing method must be invoked after the actual change (but before the commit of the transaction). For example, the after.save hook of a DAL or the after.after.save hook of a user exit script can be used.

Input:

- The parameter shown above as i.internalOrderNumber will be different per business object and per component. Also the number of parameters will differ. See the specifications for ShowAndPublishEvent in ShowAndPublishEvent on page 103.
- i.actionCode. When publishing a Process, Sync, Load, Post or Update request, the developer can set an ‘actionCode’ controlling attribute. The actionCode is used to set the actionCode XML attribute

of ActionCriteria.ActionExpression in the verb part of the BOD. The actionCode values that can be used are “Add”, “Replace”, “Change”, “Delete”, or an empty string. If empty, no actionCode attribute will be included in the BOD, and also its empty parent nodes ActionExpression and ActionCriteria will be omitted.

If i.actionCode is “Delete”, the Show method is not invoked for the object instance. In that case only the identifiers of the top-level component are included in the message to be sent. This is comparable to the ShowAndPublishStandardEvent for event action ‘delete’, see ShowAndPublishStandardEvent on page 106.

Refer to BOD Message Contents on page 152on how the action code is included in the BOD application area.

- i.documentId. Part of the BODID (see specifications below). This element must be filled.
- i.revisionId. Part of the BODID (see specifications below).
- i.variationId. Part of the BODID (see specifications below).
- i.tenantId. Part of the BODID (see specifications below). This element must be filled.
- i.accountingEntityId. Part of the BODID (see specifications below).
- i.locationId. Part of the BODID (see specifications below).

The BOD that is published will contain a BODID (refer to BOD Message Contents on page 152 on the location of the BODID in the BOD application area). The BODID will have the following format:

```
infor-nid:<tenantId>:<accountingEntityId>:<locationId>:<documentId>:<revi
sionId>?<Noun>&verb=<Verb>&variationID=<variationId>
```

For example, if:

- ShowAndPublishSyncBOD is used for the SalesOrderBOD
- i.documentId = “ 12345”
- i.revisionId = “”
- i.variationId = “24 ”
- i.tenantId =- “default”
- i.accountingEntityId = “entity”
- i.locationId = “ ”

then the BODID will be:

```
infor-nid:default:entity: :
  12345:?SalesOrder&verb=Sync&variationID=24
```

The variationID name-value pair will only be available if i.variationId is filled (not isspace).

Note that leading and trailing spaces are not removed.

Each of the IDs must be of the normalizedString type (see http://www.w3.org/TR/xmlschema-2/#normalizedString). A normalizedString must not contain carriage return (#xD), line feed (#xA) or tab (#x9) characters. The contents of the IDs are not checked inside the ShowAndPublish<Verb>BOD, except for an 'isspace' check on mandatory IDs.

It is advised to avoid colons (':') in document id, revision id variation id, tenant id, accounting entity id, and location id. If you include colons, the application receiving the event may have problems interpreting the BODID.

Output:

- Return value: 0 if successful, <> 0 if an error occurred. All errors that occur until the message is stored safely in the outgoing queue (actually a table) are reported back to the calling application.
- If the return value <> 0 then one or more DAL error messages are set. If return value = 0 then one or more DAL warning messages may be set. The DAL messages will at least include the business object name or noun and verb. The messages do not have to include all details. Detailed information may be logged in $BSE/log. In that case the message will contain a reference to the details consisting of log file name (including path) and a timestamp that is included in the log file

The component ID cannot be specified in ShowAndPublish<Verb>BOD. The component ID will always contain "erp".

When specifying incorrect input the following happens:

i.actionCode not in { "Add", "Replace", "Change", "Delete", empty string }	Return value <> 0. Note: action codes "Accepted", "Modified" or "Rejected" cannot be used in ShowAndPublish<Verb>BOD.
i.documentId is empty or contains spaces only	Return value <> 0.
Incorrect i.documentId, i.revisionId, i.variationId, i.tenandId, i.accountingEntityId, or i.locationId	This is not detected; the technology cannot validate whether the application specifies the correct values. Also no check is done on whether these parameters contain valid characters only. If one or more incorrect values are specified then ION or the receiving application may run into an error.

Other exceptions are handled in the same way as in ShowAndPublishEvent, see ShowAndPublishEvent on page 103.

Note:

- The exception handling differs significantly from ShowAndPublishEvent as used for BDE events.
- The exceptions that result in a return value <> 0 may also do so if no 'subscription' exists for the corresponding noun and verb.

The following example shows how a ShowAndPublishSyncBOD method can be used to publish events occurring in LN. The last column shows how the receiving application can act when receiving the event.

Event occurring in LN	Event Publishing	Event Processing in OnSync
Create an order	ShowAndPublishSyncBOD on header, action code 'Add'	Use the standard Create method
Change an order header	ShowAndPublishSyncBOD on header, action code 'Change'	Use the standard Change method
Delete an order	ShowAndPublishSyncBOD on header, action code 'Delete'	Use the standard Delete method

Event occurring in LN	Event Publishing	Event Processing in OnSync
Create an order line	ShowAndPublishSyncBOD on header, action code ‘Delete	Use the standard Change method after setting action type ‘createOr-Change’ for all components
	ShowAndPublishSyncBOD on header, action code ‘Replace’	Delta detection (see below)
Change an order line	same as Create an order line	same as Create an order line
Delete an order line	ShowAndPublishSyncBOD on header, action code ‘Replace’	Delta detection (see below)

Note: Unfortunately, in a BOD you cannot set action types at component level. Only an action code at message level is available. Because of this a rather complex delta detection process is needed. The main problem is that for a ‘Replace’, deleted subcomponent instances (such as order lines) are not listed explicitly in the BOD. So the receiving application must compare the incoming BOD to its own persistent data. Additionally, quite some overhead exists for unchanged component instances, because either the receiving application must check whether anything has changed, or the receiving application must simply change the existing data in the database.

For example:

Database status when re-ceiving the BOD	Incoming Sync BOD having action code ‘Replace’	Required actions to synchronize
Order X, Order line 1, Order line 2	Order X, Order line 2, Order line 3	Change Order X, Delete Order line 1, Change Order line 2, Create Order line 3

Alternatively, you can simply delete the existing order and create the new one based on the incoming BOD. However, in many cases that won’t work in practice, because deletion cannot be done if references to the order exist.

PublishEvent

The interface for PublishEvent is specified in PublishEvent on page 102. Technically, the PublishEvent interface is the same for publishing BODs.

However, when publishing an event for a BOD-type business object:

- The PublishEvent method will return errors or warnings to the application instead of publishing/logging them. This is done in the result output parameter for errors, and in the information area of the response output parameter for warnings.
- The eventAction must be a supported BOD verb.
- You can include BOD-specific controlling data.

The following BOD-specific controlling elements can be used:

- actionCode: see i.actionCode in ShowAndPublish<Verb>BOD. Note: if i.actionCode is “Replace”, the complete object (including all subcomponent instances) must be included in the data area.

- documentID: see i.documentId in ShowAndPublish<Verb>BOD..
- revisionID: see i.revisionId in ShowAndPublish<Verb>BOD..
- variationID: see i.variationId in ShowAndPublish<Verb>BOD..
- tenantID: see i.tenantId in ShowAndPublish<Verb>BOD..
- accountingEntityID: see i.accountingEntityId in ShowAndPublish<Verb>BOD..
- locationID: see i.locationId in ShowAndPublish<Verb>BOD..

For example:

```
<PublishEventRequest>
 <ControlArea>
  <eventAction>Process</eventAction>
  <actionCode>Add</actionCode>
  <documentID>12345</documentID>
  <revisionID/>
  <variationID>2</variationID>
  <tenantID>default</tenantID>
 </ControlArea>
 <DataArea>
  <PurchaseOrderBOD>
   …
  </PurchaseOrderBOD>
 </DataArea>
</PublishEventRequest>
```

The elements in the control area are handled as follows:

- The tags are handled case-insensitively.
- A warning is returned in the response if an unknown tag exists in the control area. A warning may not be returned if no subscription exists. Known tags are the following standard event tags (eventAction, eventPriority, eventTimestamp, eventTimestampSequenceNr, eventConsumer, eventHasMoreBatches, EventReference, and destination) and the BOD-specific tags (such as actionCode, organizationalPath, documentID, revisionID, and variationID), and the BOD-specific tags (such as actionCode, documentID, revisionID, and variationID).

When specifying incorrect input the following happens:

- Any of the elements occurs two or more times: this is not reported; one of the values will simply be used; probably the last one.
- No or an empty value for tenantID or documentID: an error is reported.
- Empty value for any of the other elements: use the empty value.
- Incorrect actionCode: this is not verified. The message is simply published using that actionCode; if it is really a problem then ION or the receiving application will report an error.
- Incorrect value or illegal characters used in documentID, revisionID, variationID, tenantID, accountingEntityID, or locationID: this is not verified. The message is simply published; if it is really a problem then ION or the receiving application will report an error.

‘Subscription’ Checking

When invoking one of the ShowAndPublish methods or PublishEvent, one of the first things that is done is a check on whether the BOD (or BDE event) must be published (a 'subscription' exists).

However, in some cases extra application logic is executed for publishing, even before invoking the publishing method. In that case this logic can be avoided by first checking whether publishing is on.

For that reason The following protected methods is available for BOD business objects:

```
boolean ppmmm.bl999st00.<BusinessObjectName>.EventMustBePublished (
  const     string   i.verb)
```

BOD Message Contents

Overview

A BOD consists of three parts: application area, verb, and noun. The application area and verb part are built in the Enterprise Server technology, which uses information from the BID/BII and from the ShowAndPublish<Verb>BOD parameters (or PublishEvent input) to do this. The noun part (except for the 'BOD' noun) is defined in the BID in LN Studio and the BII makes sure the correct implementation for each of the noun elements is available. Adapter logic in ION may fine-tune the contents of the BOD (including the noun area).

The following XML illustrates the structure and contents of BODs published from LN. Tags and data that will vary are shown in italics.

```
<VerbNoun
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schema.infor.com/InforOAGIS/2
   http://schema.infor.com/InforOAGIS/2/BODs/Developer/VerbNoun.xsd"
  xmlns="http://schema.infor.com/InforOAGIS/2"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  releaseID="2">
 <ApplicationArea>
  <Sender>
   <LogicalID>lid://some.logical.id</LogicalID>
   <ComponentID>erp</ComponentID>
   <ConfirmationCode>OnError</ConfirmationCode>
  </Sender>
  <CreationDateTime>2008-06-12T21:45:50Z</CreationDateTime>
  <BODID>infor-nid:tenant:accEntity:locID:docID:revID?Noun&verb=Verb&
variationID=varID</BODID>
 </ApplicationArea>
 <DataArea>
  <Verb>
   <TenantID>tenant</TenantID>
   <AccountingEntityID>accEntity</AccountingEntityID>
```

```
   <LocationID>locID</LocationID>
   <ActionCriteria>
    <ActionExpression actionCode="actionCode"/>
   </ActionCriteria>
  </Verb>
  <Noun>
   …
  </Noun>
</VerbNoun>
```

Application Area Contents

The following elements are included in the application area:

- Sender.LogicalID, to identify the sending application. The logical ID of the application instance is prefixed by ‘lid://’.

  For example: ‘lid://infor.erpenterprise.europe.001’.
- Sender.ComponentID, to identify a subcomponent within the sending application, if needed. For LN this will always be ‘erp’.
- BODID, to identify the BOD message. The BODID will differ slightly for reply BODs. Additionally, the variationID name-value pair may be omitted. For details, see below.
- ConfirmationCode. This will be ‘OnError’.
- DateTime. This is a timestamp near the end of the application database transaction that published the BOD.

The Sender.ReferenceID and other application area elements are unused.

For a request BOD the BODID will have the following format:

```
infor-nid:<tenantId>:<accountingEntityId>:<locationId>:<documentId>:<revi
sionId>
?<Noun>&verb=<Verb>&variationID=<variationID>
```

The variation ID name-value pair will only be included if it is specified. In other words, if not isspace(i.variationId). The sequence of the name-value pairs is undefined.

For example, if ShowAndPublishSyncBOD is used for the SalesOrderBOD business object, where

- i.documentId = " 12345 "
- i.revisionId = ""
- i.variationId = "24"
- i.tenantId = "default"
- i.accountingEntityId = "entity"
- i.locationId = " "

then the BODID will be:

```
infor-nid:default:entity: : 12345
  :?SalesOrder&verb=Sync&variationID=24
```

For a response BOD (Confirm, Acknowledge, and in the future also Show), the BODID is derived from the BODID from the corresponding request BOD. Only the verb will be changed and a sequence name-value pair will be added.

For example, if a BOD, having the following BODID, arrives:

```
infor-nid:default:entity:1:9876:1?SalesOrder&verb=Process&variationID=24
```

then the BODID for the response BOD will be:

```
infor-nid:default:entity:1:9876:1?SalesOrder&verb=Acknowledge&variation
ID=24&
  sequence=1
```

or, if an error occurred:

```
infor-nid:default:entity:1:9876:1?SalesOrder&verb=Confirm&variationID=24&se
quence=1
```

Verb Area

The verb area includes the tenant, accounting entity and location IDs, and the actionCode.

For reply BODs, ActionCriteria will be ResponseCriteria, and ActionExpression will be ResponseExpression. Additionally, the verb area for a reply BOD includes the OriginalApplicationArea from the corresponding request BOD.

Noun Area

The noun content is determined by the business interface as modeled in the LN Studio. The application code and BII must make sure that the IDs as included in the noun part match the IDs as specified in the ShowAndPublish<Verb>BOD method.

ID Sequence for Incoming BODs

An incoming BOD can contain multiple IDs: a universal ID and IDs used by individual applications. Before a BOD is offered to the business interface implementation in LN, the technology layer ensures that the relevant ID for the receiving LN fortress is listed first. The business interface implementation only has to check the first ID. This action is done for all sets of IDs included in the BOD.

More specifically, if multiple elements having name 'ID' are listed sequentially (without other nodes in between), then the ID having an attribute schemeAgencyID equal to i.logical.id will be moved if necessary, it is the first one in the sequence.

(Aside: actually the noun part of a BOD is responsibility of the business interface implementation / application. This method is an exception to the rule, to reduce the development efforts needed to create a business interface implementation in the LN Studio.)

For example, if the logical ID for the receiving LN application instance is "infor.ln.2" and incoming BOD contains:

```
  ...
  <PartyID>
    <AnotherElement1>value</AnotherElement1>
    <ID accountingEntity="ae" location="loc" variationID="2">X123</ID>
    <ID schemeName="System" schemeAgencyID="infor.ln.1">Y456</ID>
    <ID schemeName="System" schemeAgencyID="infor.ln.2">Z456</ID>
    <ID schemeName="System" schemeAgencyID="infor.sl.1">Q789</ID>
    <AnotherElement2>value</AnotherElement2>
  </PartyID>
  ...
```

then the BOD will be changed to:

```
...
  <PartyID>
    <AnotherElement1>value</AnotherElement1>
    <ID schemeName="System" schemeAgencyID="infor.ln.2">Z456</ID>
    <ID accountingEntity="ae" location="loc" variationID="2">X123</ID>
    <ID schemeName="System" schemeAgencyID="infor.ln.1">Y456</ID>
    <ID schemeName="System" schemeAgencyID="infor.sl.1">Q789</ID>
    <AnotherElement2>value</AnotherElement2>
  </PartyID>
  ...
```

If the BOD does not contain an ID having schemeAgencyID equal to the logical ID then the BOD will be unchanged. For example, if the logical ID is "infor.ln.2" and incoming BOD contains:

```
...
  <PartyID>
    <ID schemeName="System" schemeAgencyID="infor.sl.1">Q789</ID>
    <ID schemeName="System" schemeAgencyID="infor.ln.1">Z456</ID>
    <ID accountingEntity="ae" location="loc" variationID="2">X123</ID>
  </PartyID>
  ...
```

then the BOD will not be changed.

Using Batch Settings in Outgoing BODs

For each business interface implementation of type 'bod' that is able to publish the Sync verb, it must be possible to specify a batch id, batch sequence and batch size.

In that case the Sync BOD is published as usual, but additionally the following name-value pairs are added to the BODID of the published BOD:

batchSequence=<i.batchSequence>&batchID=<i.batchID>&batchSize=<i.batchSize>

When using 'batch', batch id must be filled, batch sequence must always be filled. Batch size can be omitted; it must be possible to publish a BOD without a batchSize name-value pair: batchSequence=<i.batchSequence>&batchID=<i.batchID>.

Handling batch-elements for incoming BODs is out of scope

Specifications

For all ShowAndPublish<Verb>BOD() methods, a new optional parameter is added:

```
long ppmmm.bl999st00.OrderBOD.ShowAndPublish<Verb>BOD(
  domain ppmmm.orno i.internalOrderNumber,  | for example
 const  string  i.actionCode,
 const  string  i.documentId,
 const  string  i.revisionId,
 const  string  i.variationId,
 const  string  i.tenantId,
 const  string  i.accountingEntityId,
 const  string  i.locationId,
  [ long  i.settings ] )
```

Parameter i.settings optionally contains a 'struct' (XML) to specify the new optional settings and any future additions.

These elements are currently handled:

- batchID
- batchSequence
- batchSize

For example:

```
<Settings>
 <batchSequence>...</batchSequence>
 <batchID>...</batchID>
 <batchSize>...</batchSize>
</Settings>
```

The behavior is as follows.

If batch elements are set, the BOD is published as usual, but additionally some name-value pairs are added to the BODID of the published BOD:

- If a batchSize element is included in the settings: batchSequence=<batchSequence.data>&batchID=<batchID.data>&batchSize= <batchSize.data>

For example: infor-nid:acme:100:1:PO123:4?PurchaseOrder&verb=Sync&batchSequence=1&batchID=infor.ln.1:1&batchSize=2

- If no batchSize element is included in the settings: batchSequence=<batchSequence.data>&batchID=<batchID.data> For example: infor-nid:acme:100:1:PO123:4?PurchaseOrder&verb=Sync&batchSequence=2 &batchID=infor.ln.1:1

If a variationID is used, then the batch elements are added after variationID name-value pair.

For example:

```
infor-nid:acme:100:1:PO123:4?PurchaseOrder&verb=Sync&variationID=1& batchSequence=2&batchID=infor.ln.1:1
```

Exception handling:

- If batchSequence, batchID or batchSize does not exist or is empty then it is not included in the BODID. Because empty values are ignored, you cannot set an empty value such as batchSequence=2&batchID=&batchSize=2.
- If batchSequence does not exist or is empty then batchID and batchSize are also ignored.
- When including the same element multiple times, only one will be used.
- For unknown elements a warning message is set. These elements are ignored, but the publishing continues, to ensure compatibility for future releases. Note that element names are case-sensitive.
- The contents of the elements is not checked; it can be checked in the application (e.g. tcboddll0001) if needed.

Publishing Get BODs and Receiving Show BODs in LN

To publish a Get BOD, LN invokes the PublishEvent for a BOD implementation. The PublishEvent request is translated to a Get BOD and sent to the bus. At some point in time, a Show BOD is received in reply to the Get BOD. The Show BOD is translated to a BDE request and the OnShow method is invoked

Exception Handling

For LN the normal exception handling applies. If Process results in a ConfirmBOD, then an Acknowledge ('Rejected') is sent to notify the application that sent the Process. If a Get results in a ConfirmBOD, no similar approach is used. The sending application is not notified when the Get results in a ConfirmBOD.

Example

PublishEvent

```
<PublishEventRequest>
    <ControlArea>
        <eventAction>Get</eventAction>
        <documentID>SKU2307TNTCP1:1</documentID>
        <revisionID></revisionID>
        <variationID></variationID>
        <tenantID>INFOR</tenantID>
        <accountingEntityID>FP7-642</accountingEntityID>
        <locationID>W_WMS1</locationID>
        <maxItemsForGet>1</maxItemsForGet>
        <ExpressionForGet>SKU2307TNTCP1</ExpressionForGet>
        <expressionLanguageForGet>InforItemID</expressionLanguageForGet>
    </ControlArea>
    <DataArea>
        <InventoryCountBOD/>
    </DataArea>
</PublishEventRequest>
```

Note:

- The actionCode cannot be used for Get.
- The tags are handled case-insensitively.

Resulting Get BOD

(Note: the name spaces are subject to change)

```
<GetInventoryCount xmlns:ns1="http://schema.infor.com/InforOAGIS/2/unit
code/66411:2001"
    xmlns<GetInventoryCount xmlns:ns1="http://schema.infor.com/In
forOAGIS/2/unitcode/66411:2001"
    xmlns:ns2="http://schema.infor.com/InforOAGIS/2/currency
code/54217:2001"
    xmlns="http://schema.infor.com/InforOAGIS/2"
   xmlns:ns4="http://schema.infor.com/InforOAGIS/2/languagecode/5639:1988"

    xmlns:ns5="http://www.openapplications.org/oagis/9/codelists"
    xmlns:ns6="http://www.openapplications.org/oagis/9/unqualified
datatypes/1.1"
    xmlns:ns7="http://www.openapplications.org/oagis/9/qualified
datatypes/1.1"
    xmlns:ns8="http://schema.infor.com/InforOAGIS/2/IANAMIMEMedi
aTypes:2003"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schema.infor.com/InforOAGIS/2 http://core
dev.infor.com/svn/BODs/branches/v2.3.x/InforOAGIS/BODs/Developer/GetInven
toryCount.xsd"
    releaseID="releaseID1" versionID="versionID1" systemEnvironment
Code="Production"
    languageCode="en-US">
    <ApplicationArea>
```

```
        <Sender>
            <LogicalID>lid://infor.ln.local</LogicalID>
            <ComponentID>ERP LN</ComponentID>
        </Sender>
        <CreationDateTime>2010-06-22T12:36:37Z</CreationDateTime>
        <BODID>infor-nid:INFOR:FP7-642:W_WMS1:SKU2307TNTCP1:1?Invento
ryCount&amp;amp;verb=Get</BODID>
:ns2="http://schema.infor.com/InforOAGIS/2/currencycode/54217:2001"
    xmlns="http://schema.infor.com/InforOAGIS/2"
   xmlns:ns4="http://schema.infor.com/InforOAGIS/2/languagecode/5639:1988"

    xmlns:ns5="http://www.openapplications.org/oagis/9/codelists"
    xmlns:ns6="http://www.openapplications.org/oagis/9/unqualified
datatypes/1.1"
    xmlns:ns7="http://www.openapplications.org/oagis/9/qualified
datatypes/1.1"
    xmlns:ns8="http://schema.infor.com/InforOAGIS/2/IANAMIMEMedi
aTypes:2003"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schema.infor.com/InforOAGIS/2 http://core
dev.infor.com/svn/BODs/branches/v2.3.x/InforOAGIS/BODs/Developer/GetInven
toryCount.xsd"
    releaseID="releaseID1" versionID="versionID1" systemEnvironment
Code="Production"
    languageCode="en-US">
    <ApplicationArea>
        <Sender>
            <LogicalID>lid://infor.ln.local</LogicalID>
            <ComponentID>ERP LN</ComponentID>
        </Sender>
        <CreationDateTime>2010-06-22T12:36:37Z</CreationDateTime>
        <BODID>infor-nid:INFOR:FP7-642:W_WMS1:SKU2307TNTCP1:1?Invento
ryCount&amp;amp;verb=Get</BODID>    </ApplicationArea>
    <DataArea>
        <Get maxItems="1">
            <TenantID>INFOR</TenantID>
            <AccountingEntityID>FP7-642</AccountingEntityID>
            <LocationID>W_WMS1</LocationID>
          <Expression expressionLanguage="InforItemID">SKU2307TNTCP1</Ex
pression>
        </Get>
        <InventoryCount>
        </InventoryCount>
    </DataArea>
</GetInventoryCount>
```

Incoming Show BOD

```
<ShowInventoryCount xmlns="http://schema.infor.com/InforOAGIS/2" relea
seID="2.3.2" versionID="2.3.2">
     <ApplicationArea>
          <Sender>
               <LogicalID>lid://infor.ln.local</LogicalID>
               <ComponentID>ERP LN</ComponentID>
          </Sender>
```

```
          <CreationDateTime>2010-06-22T21:01:02Z</CreationDateTime>
         <BODID>infor-nid:INFOR:FP7-642:W_WMS1:SKU2307TNTCP1_9:1?Invento
ryCount&amp;verb=Show</BODID>
     </ApplicationArea>
     <DataArea>
        <Show recordSetCount="1" recordSetTotal="1" recordSetCompleteIndi
cator="true">
               <TenantID>INFOR</TenantID>
               <AccountingEntityID>FP7-642</AccountingEntityID>
               <LocationID>W_WMS1</LocationID>
          </Show>
          <InventoryCount>
               <InventoryCountHeader>
                    <DocumentID>
                        <ID accountingEntity="FP7-642" location="W_WMS1"
 variationID="1">SKU2307TNTCP1_9</ID>
                    </DocumentID>
                    <DocumentDateTime>2010-06-07T20:00:39Z</DocumentDate
Time>
                    <ItemID>
                         <ID accountingEntity="FP7-642">GLB
SKU2307TNTCP1</ID>
                    </ItemID>
                    <WarehouseLocation>
                         <ID accountingEntity="FP7-642">W_WMS1</ID>
                    </WarehouseLocation>
                    <TotalQuantity unitCode="EA">3050</TotalQuantity>
               </InventoryCountHeader>
               <InventoryCountLine>
                    <LineNumber>1</LineNumber>
                    <Item>
                         <ItemID>
                              <ID accountingEntity="FP7-642">GLB
SKU2307TNTCP1</ID>
                         </ItemID>
                    </Item>
                    <Quantity unitCode="EA">1000.0</Quantity>
                    <StorageLocation>
                         <IDs>
                             <ID accountingEntity="FP7-642" sequence="1"
 sequenceName="Warehouse">W_WMS1</ID>
                         </IDs>
                    </StorageLocation>
                    <HoldCodes>
                         <Code listID="HoldCode">PIHOLD</Code>
                    </HoldCodes>
               </InventoryCountLine>
               <InventoryCountLine>
                    <LineNumber>2</LineNumber>
                    <Item>
                         <ItemID>
                              <ID accountingEntity="FP7-642">GLB
SKU2307TNTCP1</ID>
                         </ItemID>
                    </Item>
                    <Quantity unitCode="EA">1022.0</Quantity>
```

```
                    <HoldCodes>
                         <Code listID="HoldCode">PIHOLD</Code>
                    </HoldCodes>
                </InventoryCountLine>
            </InventoryCount>
        </DataArea>
</ShowInventoryCount>
```

Note: The following elements can be used and may become available in the control data:

- · recordSetCount: This is the sequence number of this BOD when a series of BODs is used as a response to a single Get.
- · recordSetTotal: This optional element contains the anticipated number of BODs that will be sent in this batch of Show messages. It may not be completely accurate, and should be treated as an estimate. This can be used by a progress bar to show percent complete.
- recordSetCompleteIndicator: This is used to indicate the last BOD in a sequence.

Corresponding Request for the OnShow Method

```
<OnShowRequest>
    <ControlArea>
        <BODApplicationArea>
            <Sender>
                <LogicalID>lid://infor.ln.local</LogicalID>
                <ComponentID>ERP LN</ComponentID>
            </Sender>
            <CreationDateTime>2010-06-22T21:01:02Z</CreationDateTime>
           <BODID>infor-nid:INFOR:FP7-642:W_WMS1:SKU2307TNTCP1_9:1?Inven
toryCount&amp;verb=Show</BODID>
        </BODApplicationArea>
        <BODDataArea>
            <Show recordSetCompleteIndicator="true">
                <TenantID>INFOR</TenantID>
                <AccountingEntityID>FP7-642</AccountingEntityID>
                <LocationID>W_WMS1</LocationID>
            </Show>
        </BODDataArea>
        <processingScope>NA</processingScope>
    </ControlArea>
    <DataArea>
        <InventoryCountBOD>
            <InventoryCountHeader>
                <DocumentID>
                    <ID accountingEntity="FP7-642" location="W_WMS1"
variationID="1">SKU2307TNTCP1_9</ID>
                </DocumentID>
               <DocumentDateTime>2010-06-07T20:00:39Z</DocumentDateTime>

                <ItemID>
                    <ID accountingEntity="FP7-642">GLBSKU2307TNTCP1</ID>
                </ItemID>
                <WarehouseLocation>
                    <ID accountingEntity="FP7-642">W_WMS1</ID>
                </WarehouseLocation>
                <TotalQuantity
```

```
                    unitCode="EA">3050</TotalQuantity>
            </InventoryCountHeader>
            <InventoryCountLine>
                <LineNumber>1</LineNumber>
                <Item>
                    <ItemID>
                        <ID accountingEntity="FP7-642">GLB
SKU2307TNTCP1</ID>
                    </ItemID>
                </Item>
                <Quantity unitCode="EA">1000.0</Quantity>
                <StorageLocation>
                    <IDs>
                      <ID accountingEntity="FP7-642" sequence="1" sequen
ceName="Warehouse">W_WMS1</ID>
                    </IDs>
                </StorageLocation>
                <HoldCodes>
                    <Code listID="HoldCode">PIHOLD</Code>
                </HoldCodes>
            </InventoryCountLine>
            <InventoryCountLine>
                <LineNumber>2</LineNumber>
                <Item>
                    <ItemID>
                        <ID accountingEntity="FP7-642">GLB
SKU2307TNTCP1</ID>
                    </ItemID>
                </Item>
                <Quantity
                    unitCode="EA">1022.0</Quantity>
                <HoldCodes>
                    <Code listID="HoldCode">PIHOLD</Code>
                </HoldCodes>
            </InventoryCountLine>
        </InventoryCountBOD>
    </DataArea>
</OnShowRequest>
```

Name Space in BODs Published from LN

The main node for BODs that used to be published fromLN contained these XML attributes:

```
<SyncSalesOrder
  xmlns="http://schema.infor.com/InforOAGIS/2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schema.infor.com/Trunk/InforOAGIS
  http://schema.infor.com/Trunk/InforOAGIS/BODs/Developer/SyncSalesOr
der.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  releaseID="2">
```

The values for xmlns, xsi:schemaLocation and releaseID were fixed when they were developed (September 2009). This can be a problem if they are changing:

1. The values will change with a new schema version of the BOD.
2. During development other values can be used (for example, a reference to a 'Trunk' location) than in the final release.
3. The standards have unfortunately appeared to be volatile.
4. Trunk' does not contain a specific schema version. The contents of 'Trunk' can change in the future.

Non-event driven BODs

Non-event driven BODs are used for initial load of the applications that receive the BODs. With the situation prior to the introduction of non-event driven BODs, Sync BODs can be used to update applications. A disadvantage is that all subscribers will receive these BODs resulting in network load and unwanted Confirm BODs for subscribers that are already up-to-date.

A non-event driven BOD is a Show BOD that will be sent to the applications specified by their Logical ID. If a BOD should be sent to all subscribers, the Sync BOD could be used.

Note that if a new subscriber is updated with initial load, it can happen that at the same time a Sync BOD is published. The Sync BOD can contain an update on a not yet received instance. This will be processed as an Add.

Adaptions in LN Studio

In the LN Studio modeling environment it should be possible to model the Show event without a Get. The current constraint checks should be removed. Also, the ShowAndPublishShowBOD method should be generated in case "can be published" is enabled for the Show event.

The specification of the ShowAndPublishShowBOD is:

```
function extern long ppmmmstccc00.COMPONENT.ShowAndPublishShowBOD(
 ... (identifiers)
 const string i.actionCode, | action code, normally 'Add' for Show BODs
 const string i.documentId, | the document ID
 const string i.revisionId, | the revision ID
 const string i.variationId, | the variation ID
 const string i.tenantId, | the tenant ID
 const string i.accountingEntityId, | the accounting entity ID
 const string i.locationId, | the location ID
 [ long i.settings ] ) | the settings (see below)
```

Returns 0 (OK) or DALHOOKERROR (Error).

i.settings contains an XML with all To Logical IDs, in this format:

```
<Settings>
 <ToLogicalId>...</ToLogicalId> (one or more instances)
```

```
 ...
 ... (may include other settings as well)
</Settings>
```

Adaptions in LN / Baan IV/ Baan 5.0

Outgoing Show BODs having To Logical IDs are supported. When a BOD is written to the outbox, the To Logical ID must be specified for the Show BOD. If a BOD is sent to multiple logical IDs, it is duplicated by writing it to the outbox for each To Logical ID separately.

Synchronous execution in PublishEvent

For processing of a BOD, a PublishEvent is carried out in a separate BDE. This function runs in a separate process, but has its own transaction context and runs asynchronous. If an error occurs, it is not reported.

The PublishEvent call can be configured such that the processing can be made synchronous (i.e. further processing is suspended until PublishEvent is finished), running in the same transaction context (i.e. uncommitted data from the parent process can be read and the data is committed when the parent process commits), and errors can be caught in a Result message.

To enable these features the following must be put in the ControlArea of the Request:

```
<ControlArea synchronousCall="true" sameTransaction="true" getResult="true">
```

This will only work if the ControlArea also has a subnode 'localDestinationCompany'.

The attributes mean:

- synchronousCall=true: the call is synchronous
- sameTransaction=true: the subprocess uses the transaction context of the current process
- getResult=true: in case of errors, they are not published but returned in a Result node (only in combination with synchronousCall=true).

(by default all these parameters are false).

Chapter 10: Using the LN Studio for Baan IV

Overview

In LN Studio, you can develop business interfaces for Baan IV.

This requires the following:

- Specific Baan IV solutions.
- A minimum version of the Baan IV Porting Set.
- A minimum version of the LN Studio.

The prerequisites are documented in solution 22829846.

Limitations

Compared to the usage for LN, the usage of the LN Studio for Baan IV has these limitations:

- You cannot use related software projects.
- The LN Studio for Baan IV is intended for the development of BOD business interfaces. The BDE interface is only supported for protected implementations.
- Baan IV does not store meta data for business objects. Therefore, you cannot use the BI importer and the WSDL importer.
- BDE event publishing is unavailable. Therefore, you cannot use the following methods with Baan IV:
  - PublishEvent
  - OnEvent
  - SubscribeEvent
  - UnsubscribeEvent
  - SubscribeList.
- You can use function server implementations. However, the Form Importer is not (yet) available for Baan IV. Therefore, you cannot import forms from Baan IV into the LN Studio.
- You cannot use referential integrity methods (CreateRef, DeleteRef) on Baan IV.

How to use the LN Studio for Baan IV

In general, creating BIDs and BIIs for Baan IV is similar to creating BIDs and BIIs for LN. However, take into account that Baan IV technology differs from LN technology.

For example:

- No DALs are available. Therefore, ensure all required business logic is included in BII hooks or in libraries that are invoked from BII hooks.
- Data types differ. For example, in Baan IV the UTC data type is unavailable. Separate date and time columns are used instead of a single UTC column. To map UTC attributes to date and time columns, create ‘on set’ and ‘on get’ hooks.
- There is no single DAL script that encapsulates the data for a table. Therefore, you probably must invoke the ShowAndPublish<Verb>BOD from multiple locations in the application.

In Enterprise Server, you can view existing business objects in the **Business Objects (ttadv7500m000)** session. In Baan IV this session is unavailable. To delete or expire generated business objects, you must use LN Studio.