

VisualAge Pacbase



**eBusiness & Pacbench C/S Applications**  
**Proxy Programming Interface**  
*Version 3.5*



Note

Before using this document, read the general information under 'Notices' on the next page.

You may consult or download the complete up-to-date collection of the VisualAge Pacbase documentation from the VisualAge Pacbase Support Center at:

<http://www.ibm.com/software/awdtools/vapacbase/productinfo.htm>

Consult the Catalog section in the Documentation home page to make sure you have the most recent edition of this document.

**First Edition (June 2004)**

This edition applies to the following licensed program:

VisualAge Pacbase Version 3.5

Comments on publications (including document reference number) should be sent electronically through the Support Center Web site at:

<http://www.ibm.com/software/awdtools/vapacbase/support.htm>

or to the following postal address:

IBM Paris Laboratory

VisualAge Pacbase Support

1 place J.B. Clément

93881 Noisy-le-Grand Cedex France

© Copyright International Business Machines Corporation 1983,2004. All rights reserved.

---

# Summary

A detailed *Table of Contents* is presented in the following pages.

<b>Notices</b> .....	<b>ix</b>	Actions Performed on a Remote Server.....	66
<b>Trademarks</b> .....	<b>xi</b>	Externalization of the Management of Requests.....	83
<b>Foreword</b> .....	<b>1</b>	XML flows handling .....	84
<b>Chapter 1. Classes</b> .....	<b>3</b>	Management of Proxy Context.....	87
Data Classes .....	3	<b>Chapter 4. Events</b> .....	<b>89</b>
ProxyLv Class: Inheritance Diagrams .....	5	Management of Paging.....	89
Classes of XML flows handling .....	5	Management of Unit Reads .....	90
<b>Chapter 2. Attributes</b> .....	<b>7</b>	Management of Simultaneous Selections .....	91
Management of Selections .....	7	Management of Logical Locks.....	91
Management of Updates.....	11	Management of Dependent Instances.....	92
Exchange Flow Check .....	13	<b>Chapter 5. Public Interface for Data Elements Handling</b> .....	<b>93</b>
Logical View Instance Container.....	15	Management of a Data Element's Contents .....	93
Management of User Services.....	24	Management of Authorized-Value Codes.....	93
Management of Logical Locks .....	25	Management of Authorized Values .....	94
Management of Selection-Return Messages .....	26	Management of the Validity of a Data Element's Contents.....	94
Management of Events.....	27	Access to the Characteristics of a Data Element .....	95
Management of Contextual Information.....	28	Initialization of the Data Elements Values .....	95
Available Counters .....	29	Management of a Data Element's Presence.....	96
Management of Communications .....	31	Management of a Data Element's Check .....	96
Management of Asynchronous Conversations .....	40	Management of membership to a Sub-schema.....	97
Conversation Time.....	42	Management of membership to an extraction method .....	97
Sub-Schema Management .....	43	<b>Chapter 6. Management of Errors</b> .....	<b>99</b>
External Request Management .....	44	Management of Errors for Java Target.....	99
Use of a JTable .....	45	Management of Errors for COM Target .....	104
<b>Chapter 3. Actions</b> .....	<b>47</b>	<b>Index</b> .....	<b>111</b>
Actions Performed Locally .....	47		



# Table of Contents

<b>Notices.....</b>	<b>ix</b>
---------------------	-----------

<b>Trademarks .....</b>	<b>xi</b>
-------------------------	-----------

<b>Foreword .....</b>	<b>1</b>
-----------------------	----------

<b>Chapter 1. Classes .....</b>	<b>3</b>
---------------------------------	----------

Data Classes .....	3
Inheritance Diagrams .....	3
Java and COM .....	3
DataDescription Class.....	3
SelectionCriteria Class .....	4
DataDescriptionUpdate Class.....	4
UserContext Class.....	4
ProxyLv Class: Inheritance Diagrams .....	5
Java and COM .....	5
Classes of XML flows handling .....	5
Inheritance Diagrams .....	5
Java and COM .....	5
Generic classes.....	6
Generated classes.....	6

<b>Chapter 2. Attributes .....</b>	<b>7</b>
------------------------------------	----------

Management of Selections.....	7
Selection Criteria.....	7
List of Available Extraction Methods .....	8
Extraction Method to be Executed.....	9
Management mode of the collection.....	10
Management of Updates.....	11
Implementation of Server Data Checks.....	11
Server Data Refresh.....	12
Exchange Flow Check .....	13
Limited Number of Exchanged Instances.....	13
Unlimited Number of Exchanged Instances .....	14
Logical View Instance Container.....	15
Instance List Presentation.....	15
Selection of Local or Server Criterion for Instance List Sort .....	16
Local Criterion for Instance List Sort.....	17
Instance Presentation .....	18
Presentation of Modified Folders.....	19
Presentation of Modified Instances.....	20
Presentation of Instances for a User Service.....	21
Presentation of Instances Returned by a User Service.....	22
Presentation of an Instance Linked to a User Service .....	23
Management of User Services.....	24
List of Available User Services .....	24
User Service to be Executed .....	24
Management of Logical Locks .....	25
Folder Lock Identifier.....	25
Management of Selection-Return Messages.....	26
Message Label .....	26

Message Key.....	26
Management of Events .....	27
List of Events from the Last Server Action.....	27
Management of Contextual Information .....	28
Contextual Information .....	28
Contextual Information Associated with Reference Nodes .....	28
Available Counters.....	29
Total Number of Local Instances .....	29
Total Number of Update Services .....	29
Number of Update Services Associated with a Node .....	30
Management of Communications.....	31
List of Available Platforms .....	31
Platform Selected for a Query Execution .....	32
Log-in User Code .....	32
Log-in Password.....	33
Name of the Machine Hosting the Java/VisualAge Pacbase Gateway .....	33
IP Port Associated with the Communications Manager .....	34
Setting of the Platforms File's Address .....	34
Selection of the Communication Adapter .....	35
Management of Communication Parameters.....	36
Management of Asynchronous Conversations.....	40
Determining the Type of Conversation .....	40
Last Identifier of an Asynchronous Conversation .....	40
Maximum Number of Pending Replies.....	41
Number of Pending Replies .....	41
Conversation Time .....	42
Communication Time .....	42
Execution Time of the Server Processing .....	42
Sub-Schema Management.....	43
List of Available Sub-Schemas .....	43
Sub-schema to be Taken into Account.....	44
External Request Management.....	44
Access and Set a Request.....	44
Use of a JTable.....	45
Display of the Instances Collection in a JTable.....	45
Display of the Updated Folders in a JTable .....	45
Display of the Updated Instances in a JTable .....	46
Display of the Instance Collection in input/output by a User Service in a JTable .....	46

<b>Chapter 3. Actions .....</b>	<b>47</b>
---------------------------------	-----------

Actions Performed Locally .....	47
Updates .....	47
Creation of a Logical View Instance.....	47
Modification of a Logical View Instance .....	48
Deletion of a Logical View Instance.....	49
Cancellation of Updates .....	50
Cancellation of a Folder's Updates.....	50
Cancellation of all Folders Updates .....	51
Cancellation of Updates on a Node Instance.....	52

Cancellation of Updates on all the Instances of a Node .....	53	Sub-Schema Management.....	82
Management of User Services.....	54	Test of Communication with the Server .....	83
Assignment of an Instance to a User Service.....	54	Externalization of the Management of Requests .....	83
Modification of an Assigned Instance .....	55	Creation of a Request.....	83
Deletion of an Assigned Instance.....	56	Execution of the Request Actions on the Server.....	84
Local Navigation in the Folders .....	57	Cancellation of the Request Actions.....	84
Current Selection of an Instance in a Folder .....	57	XML flows handling .....	84
Selection of an instance from an Index.....	58	Status of a Logical View Instance .....	85
Selection of an Instance Associated with a User Service.....	58	Obtaining of a XML flow .....	85
Reactivation of the Current Selection.....	59	Update from a XML flow .....	86
Miscellaneous Initializations.....	59	Management of Proxy Context.....	87
Initialization of the collection .....	59	Initialization of the proxy context.....	87
Initialization of Extraction Methods.....	60	Initialization of the local cache.....	88
Initialization of the User Services .....	60	Retrieval of the Proxy .....	88
Initialization of the "Presentation of Instances for a User Service" Container .....	61	<b>Chapter 4. Events.....</b>	<b>89</b>
Initialization of the Update-Refresh Option.....	61	Management of Paging.....	89
Initialization of Selection Criteria .....	62	Signal of Retrieval of a Collection's Last Page.....	89
Addition of instances in the local cache without server access .....	62	Signal of Retrieval of a Collection's First Page .....	89
Management of Referenced Instances.....	63	Signal of Presence of at Least One Following Page ....	90
Assignment of a Referenced Instance.....	63	Signal of Presence of at Least One Preceding Page ....	90
Retrieval of Proxies' Generation Contexts .....	64	Management of Unit Reads .....	90
Generation Context of a Folder .....	64	Signal of Reading of a Record not Found.....	90
Generation Context of a Node.....	65	Management of Simultaneous Selections .....	91
Sub-Schema Management .....	65	Signal of Non-Participation to a Simultaneous Read .	91
No Selection of Sub-Schema .....	65	Management of Logical Locks.....	91
Actions Performed on a Remote Server .....	66	Signal of Assigned Logical Lock .....	91
Selection on a Node .....	66	Signal of Unsuccessful Logical Lock .....	91
Selection of a Set of Instances .....	66	Management of Dependent Instances .....	92
Reading of an Instance with or without Logical Locking.....	67	Signal of Presence of at Least One Dependent Instance	92
Reading of Instances from identifiers.....	68	Signal of Absence of Dependent Instances.....	92
Concurrent Selection on Multiple Nodes with or without Locking.....	69	<b>Chapter 5. Public Interface for Data Elements Handling.....</b>	<b>93</b>
Reading of an Instance and its Immediate Hierarchy .....	69	Management of a Data Element's Contents .....	93
Reading of an Instance and its Complete Hierarchy .....	70	Management of Authorized-Value Codes.....	93
Reading of the Immediate Hierarchy of a Current Instance.....	71	Management of Authorized Values .....	94
Reading of the Complete Hierarchy of a Current Instance.....	72	Management of the Validity of a Data Element's Contents	94
Anticipated Reading of an Instance's Immediate Hierarchy .....	72	Access to the Characteristics of a Data Element.....	95
Anticipated Reading of an Instance's Complete Hierarchy .....	73	Initialization of the Data Elements Values .....	95
Management of Paging.....	73	Management of a Data Element's Presence.....	96
Reading of the Following Page's Instances.....	73	Management of a Data Element's Check .....	96
Reading of the Preceding Page's Instances.....	75	Management of membership to a Sub-schema.....	97
Sending of Updates .....	76	Data Element Belonging to the Sub-Schema.....	97
Sending of Local Updates to the Server .....	76	Management of membership to an extraction method ....	97
Management of Logical Locks.....	77	<b>Chapter 6. Management of Errors .....</b>	<b>99</b>
Logical Locking of a Current Instance .....	77	Management of Errors for Java Target.....	99
Logical Unlocking of a Current Instance .....	78	Classes Related to the Management of Errors .....	100
Management of Dependent Instances .....	79	Communication Errors.....	100
Check on the Presence of Dependent Instances...	79	System Errors.....	100
Management of User Services.....	80	Local Errors .....	100
Execution of User Services .....	80	Server Errors .....	101
Management of Asynchronous Conversations.....	81		
Deferred Retrieval of a Reply .....	81		
Check on a Message Identifier's Validity.....	82		

Error Messages Received from the Server... 101	Management of the Error Gravity ..... 105
Error Messages Received from the Server on	Events Linked to Errors..... 106
the Update ..... 101	Signal for No Error Detection ..... 106
Customizing Error Messages..... 102	Signal for Local Error Retrieval ..... 106
Local Error Messages ..... 102	Signal for Server Error Retrieval..... 106
Local Error Messages Received from the Server	Signal for System Error Retrieval ..... 106
Component ..... 102	Signal for Communication Error Retrieval ..... 107
Server and System Error Messages..... 103	Customizing Error Messages..... 107
Example of Error Messages File ..... 103	Naming Rules for Error Messages Files ..... 107
Management of Errors for COM Target..... 104	Syntax of Error Messages Files ..... 107
Access Method to Errors..... 104	Error Messages for Local Exceptions ..... 107
VapError Attributes..... 105	Local Error Messages Received from the Server
Management of the Error Type ..... 105	Component..... 108
Management of the Action Which Triggers the	Server and System Error Messages ..... 108
Error ..... 105	Example of Error Message File ..... 108
Management of the Error Key ..... 105	
Management of the Error Label ..... 105	<b>Index..... 111</b>





---

## Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to Intellectual Property and Licensing, International Business Machines Corporation, North Castle Drive, Armonk, New-York 10504-1785, USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of information which has been exchanged, should contact: IBM Paris Laboratory, SMC Department, 1 place J.B. Clément, 93881 Noisy-le-Grand Cedex, FRANCE

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM may change this publication, the product described herein, or both.



---

## Trademarks

IBM is a trademark of International Business Machines Corporation, Inc.

AIX, AS/400, CICS, CICS/MVS, CICS/VSE, COBOL/2, DB2, IMS, MQSeries, OS/2, PACBASE, RACF, RS/6000, SQL/DS, TeamConnection, and VisualAge are trademarks of International Business Machines Corporation, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

All other company, product, and service names may be trademarks of their respective owners.



---

## Foreword

This manual provides a comprehensive description of the Public Interface of components generated for the Graphic Clients of eBusiness, according to the Java and COM-compliant environments.

Each Proxy object's interface is generated using the characteristics –defined in VisualAge Pacbase– of a Logical View and its associated Elementary Component.

A Proxy object's public interface is made up of classes, characterized by a set of attributes or properties, actions or methods, and events. A GUI application handles these interface elements in order to manage each Logical View's processing according to its associated Elementary Component.

### Organization of the manual

This manual is divided into chapters, followed by an Index.

- *Chapter 1. Classes*, on page 3, describes the classes of the public interface, including inheritance trees.
- *Chapter 2. Attributes*, on page 7, provides the list of all attributes types existing for the various environments. For each attribute, the type (internal code), name (use name) and **get/set** are given.
- *Chapter 3. Actions*, on page 47, describes the actions (called methods for the Java environment) –whether local or remote– with the declaration and use name.
- *Chapter 4. Events*, on page 89, describes the Events and lists their codes.
- *Chapter 5. Public Interface for Data Elements Handling* , on page 93, documents the API for handling Data Elements.
- *Chapter 6. Management of Errors* , on page 99, describes the error management for both the Java and COM platforms.

### Prerequisites and further reading

You should be familiar with the basic principles of the eBusiness Function. The explanations given in this manual assume you have such knowledge. For detailed information about these principles, see the *eBusiness & Pacbench C/S Applications - Concepts & Architecture* guide.

If you are new to this type of development, you may find it useful to read the *eBusiness & Pacbench C/S Applications - Graphic Presentation* guide. This guide is designed to assist you in the development of Graphic Client Components, through the presentation of various examples.

## Conventions

The **courier** font signals any character string displayed or to be typed, as well as characters representing generated code.

The indication: "Internal Code" signals the code you will have to type in classic programming.

The indication: "Use Name" signals the corresponding label displayed in the Composition Editor, used in visual programming.



Cross-reference, *in italics*, to another location in this manual or in another manual. These cross-references are hyperlinks (position the mouse cursor, with a double click you will display the target of the hyperlink).

Caution, for a cross-reference to another manual, you open the home page of the Documentation on the VisualAge Pacbase Internet site, locate the concerned manual and double click.



Precaution to be taken (for risky or irreversible action...).

---

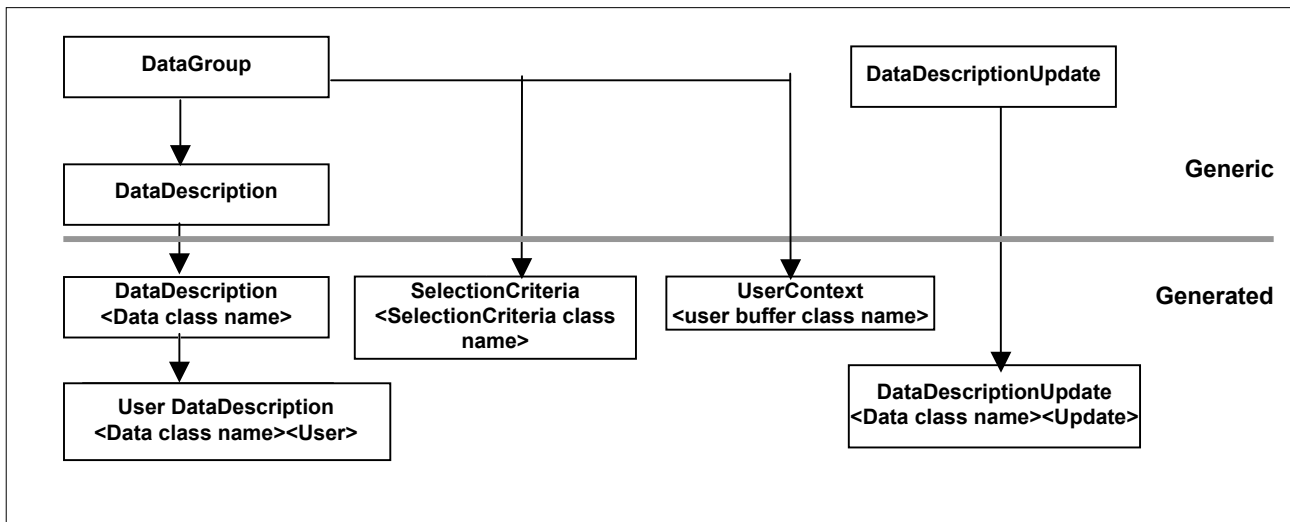
# Chapter 1. Classes

---

## Data Classes

### Inheritance Diagrams

#### Java and COM



- In PB 300 version, naming of DataDescription, SelectionCriteria and UserContext classes is made when creating Folders and the eBusiness Application.

As generating referenced nodes is optional, the naming process is made dynamics and should follow this rule:

<Name of the defined class><Folder name>

- If the generation is resulting from Folders defined in the PB 250 version, naming classes should follow this rule:

DataDescription	:	<LogicalView><Data>
User DataDescription	:	<LogicalView >>UserData>
SelectionCriteria	:	<LogicalView >>SelectionCriteria>
UserContext	:	<LogicalView >>UserContext>
DataDescriptionUpdate	:	<LogicalView >>DataUpdate>

### DataDescription Class

This class is generated for each Logical View or node of a Folder. It represents the structure of a Logical View, by defining one attribute for each identifier- or composition-type Data Element.

In the context of a Folder, the **DataDescription** class associated with a dependent node does not expose the identifier-type Data Elements of higher nodes.

An instance of this class corresponds to a Logical View instance handled by the application's graphic interface.

### **SelectionCriteria Class**

This class is generated for each Logical View or node of a Folder. It represents the structure of the key and of the extraction parameters of a Logical View, by defining one attribute for each key- or extraction-parameter-type Data Element.

In the context of a Folder, the **SelectionCriteria** class associated with a dependent node does not expose the key-type Data Elements of nodes higher in the hierarchy.

There is only one instance of this class for each Logical View or node. This instance can be used to define the identifier –and possibly the extraction parameters– of the beginning of a collection (such as SelectInstance), or of a direct read (such as ReadInstance).

### **DataDescriptionUpdate Class**

This class is generated for Folders' roots or dependent nodes, which can be modified. It represents the structure of a modified Logical View, and qualifies its modification type:

- **Created:** The Logical View associated with the node has been created locally.
- **Modified:** The Logical View associated with the node has been modified locally.
- **Deleted:** The Logical View associated with the node has been deleted locally.
- **Read:** At least one dependent instance in the Folder has been updated locally.

### **UserContext Class**

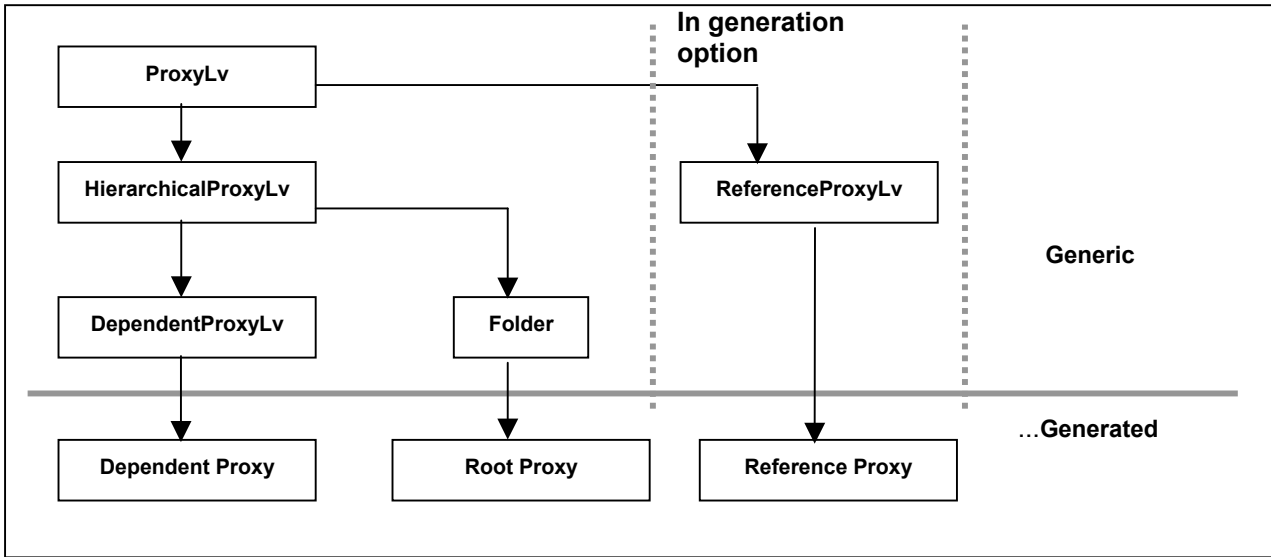
This class is generated for each eBusiness Application in which a User Buffer was defined. It represents the structure of the buffer, by defining an attribute for each Data Element.

There is only one instance of this class. It is updated by the GUI or by replies from the remote server.



# ProxyLv Class: Inheritance Diagrams

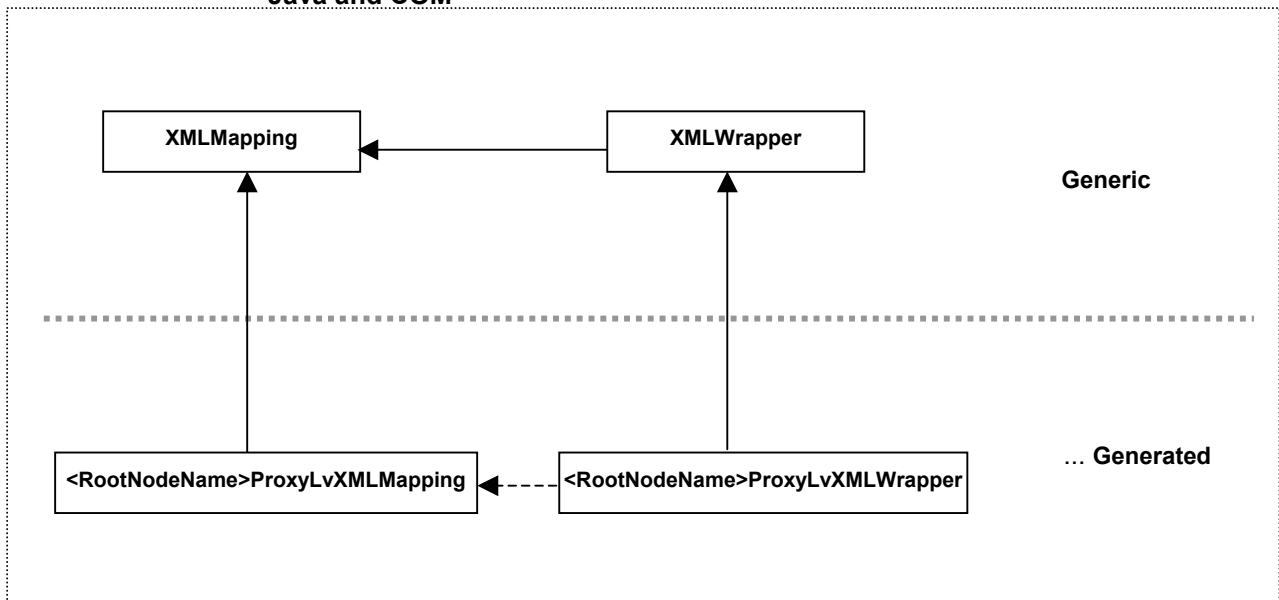
Java and COM



# Classes of XML flows handling

Inheritance Diagrams

Java and COM



At the end of the generation, an XML schema is also constituted for a folder or a folder view (coded <FolderName>.xsd or <FolderName-ViewName>.xsd). This schema, respecting the recommendations of the W3C contains the description of data.

## **Generic classes**

The XMLMapping and XMLWrapper classes define the generic API of XML flows handling.

## **Generated classes**

The XML mapping class is a XMLMapping-type singleton class. It describes the elements of the mapping specific to a folder or folder view.

The XML wrapper class is a XMLWrapper-type class associated with the generated instance of XMLMapping class. It offers methods that allow for each node to mask the request elements. This class is generated for a folder or a folder view.

---

## Chapter 2. Attributes

An attribute is linked to one of three types of elements making up the public interface of a class. For the public classes associated with a Proxy it may be a constant, a parameter or the result of an action. It is initialized by the application which uses the Proxy or by the Proxy itself, depending on the context.

---

### Management of Selections

#### Selection Criteria

##### Description

This attribute defines all the Data Elements of the key- or extraction parameter-type, defined in the Logical View associated with a node.

For dependent nodes, key-type Data Elements already defined in the same attribute of the parent node are not exposed.

The description order is that defined in the Logical View.

Each Data Element exposed is set to an “empty” value, or to its default value if it is defined in the Repository.

This attribute is always available on root-, dependent-, and reference-type nodes.

It is available for read and write access.

#### Java

- Type `{Name of generated class SelectionCriteria}`
- Internal code `selectionCriteria`
- Use name `Selection Criteria`
- get/set `public {Name of generated class SelectionCriteria} selectionCriteria() / set not available`

#### COM

- Type `{Name of generated class SelectionCriteria}`
- Internal code `selectionCriteria`
- get/set C++ `public LPDISPATCH getSelectionCriteria() / set not available`

## List of Available Extraction Methods

### Description

This attribute exposes a list of extraction-method codes defined in the Elementary Component which manages the Logical View associated with the node.

It is available on root-, dependent-, and reference-type nodes, when at least one extraction method is defined in the Elementary Component managing the Logical View associated with the node.

It is available for read-only access.

### Java

- Type `java.lang.String[]`
- Internal code `extractMethodCodes`
- Use name `Extraction Method List`
- get/set `public String[] getExtractMethodCodes ()` / set not available

### COM

This information is not identified as an attribute reference in the Proxy's API. To access to this information, follow this method:

- Internal code `getExtractMethodCodes`
- Declaration `public VapCollection getExtractMethodCodes ()`
- Nb of elements `public long getExtractMethodCodesCount ()`
- Element `public char* getExtractMethodCodesElementAt (long i)`

## Extraction Method to be Executed

### Description

This attribute defines the extraction-method code to be implemented on a collection-selection action.

It may be set to an initialization value, if this is defined in the Proxy's Settings window.

It is available on root-, dependent-, and reference-type nodes, when at least one extraction action is defined in the Elementary Component managing the Logical View associated with the node.

It is available for read and write access.

### Java

- Type `java.lang.String`
- Internal code `extractMethodCode`
- Use name `Extraction Method Code`
- get/set 

```
public String getExtractMethodCode () /
public void setExtractMethodCode (String ExtractMethodCode)
```

### COM

- Type `String`
- Internal code `extractMethodCode`
- get/set C++ 

```
public BSTR getExtractMethodCode () /
public void setExtractMethodCode (LPCTSTR s)
```

## Management mode of the collection

### Description

This attribute defines the collection management type, in return of a collection-selection action.

Two modes are available:

- Automatic management (default value)
- Manual management

The automatic management enables the replacement of the current collection with the selected instances, in return of a collection-selection action.

The manual management enables the completion of the current collection with the selected instances, in return of a collection-selection action. A selected instance, already present in the current collection, is refreshed if the instance of the current collection has not been locally modified.

The switch from one mode to another one does not lead to any change of the current collection.

As a default, the attribute is set to **false**.

In manual management, the pagination type for root- and reference-type nodes is always **extend**.

This attribute is systematically available on root-, dependent- or reference-type nodes.

It is available for read and write access.

### Java

- Type **Boolean**
- Internal code **manualCollectionReset**
- Use name **Manual Collection Reset**
- get/set **public Boolean isManualCollectionReset()  
public void setManualCollectionReset(boolean b)**

### COM

- Type **Boolean**
- Internal code **manualCollectionReset**
- get/set C++ **public BOOL getManualCollectionReset()  
public void setManualCollectionReset(BOOL b)**

---

## Management of Updates

### Implementation of Server Data Checks

#### Description

This attribute triggers the data checks on the server when a server update method is being executed.

As a default, it is set to **false**. It may be set to another initialization value, if this is defined in the Proxy's Settings window.

It is available on root- and dependent-type nodes, when the associated Logical View is designed for update mode in its Elementary Component and the **CHECKSER** option of this Elementary Component is defined.

It is available for read and write access.

#### Java

- Type **Boolean**
- Internal code **serverCheckOption**
- Use name **Server Check Option**
- get/set 

```
public Boolean getServerCheckOption()  
public void setServerCheckOption(boolean b)
```

#### COM

- Type **Boolean**
- Internal code **serverCheckOption**
- get/set C++ 

```
public BOOL getServerCheckOption()  
public void setServerCheckOption(BOOL b)
```

## Server Data Refresh

### Description

This attribute retrieves Logical View instances modified following an update performed by an Elementary Component. This function applies mainly to Logical Views with Data Elements calculated by the server.

As a default, this attribute is set to **false**. It may be set to another initialization value, if this is defined in the Proxy's Settings window.

It is available on root- and dependent-type nodes, when the associated Logical View is designed for update in its Elementary Component.

It is available for read and write access.

### Java

- Type **Boolean**
- Internal code **refreshOption**
- Use name **Refresh Option**
- get/set **public Boolean getRefreshOption()  
public void setRefreshOption(boolean b)**

### COM

- Type **Boolean**
- Internal code **refreshOption**
- get/set C++ **public BOOL getRefreshOption()  
public void setRefreshOption(BOOL b)**



---

## Exchange Flow Check

### Limited Number of Exchanged Instances

#### Description

This attribute defines the maximum number of Logical View instances returned in one exchange by an Elementary Component upon a collection-retrieving action.

It is set to the Logical View's default iterative capacity. It may be set to another initialization value, if this is defined in the Proxy's Settings window.

It may be set to a value between 0 and n, where n may exceed the Logical View's iterative capacity. When this value is 0, simultaneous-selection actions on multiple nodes do not propagate reading requests to the current node.

This attribute is available on root- or reference-type nodes, and on dependent nodes with a maximum cardinal value of n.

It is available for read and write access.

#### Java

- Type `Integer`
- Internal code `maximumNumberOfRequestedInstances`
- Use name `Maximum Number of Requested Instances`
- get/set 

```
public int getMaximumNumberOfRequestedInstances ()
public void setMaximumNumberOfRequestedInstances (int i)
```

#### COM

- Type `Integer`
- Internal code `maxNumberOfRequestedInstances`
- get/set C++ 

```
public long getMaxNumberOfRequestedInstances ()
public void setMaxNumberOfRequestedInstances (long i)
```

## Unlimited Number of Exchanged Instances

### Description

This attribute retrieves all the instances found in the database for the collection defined by the selection action. This function may generate a high number of exchanges between the Client Component and the Server Component.

As a default, it is set to **false**. It may be set to another initialization value, if this is defined in the Proxy's Settings window.

This attribute is available on root- and reference-type nodes, as well as dependent nodes with a maximum cardinal value of n.

It is available for read and write access.

### Java

- Type **Boolean**
- Internal code **globalSelection**
- Use name **Global Selection**
- get/set **public Boolean getGlobalSelection()  
public void setGlobalSelection(boolean b)**

### COM

- Type **Boolean**
- Internal code **globalSelection**
- get/set C++ **public BOOL getGlobalSelection()  
public void setGlobalSelection(BOOL b)**

---

# Logical View Instance Container

## Instance List Presentation

### Description

This attribute contains the current collection of the node to which it is associated. This collection is made of a set, or row, of Logical View instances. It results from the last reading action(s), and from local update actions performed on the node.

For a root node, the collection of instances exposed contains the Folders collection retrieved locally.

For a dependent node, the collection of instances exposed depends on the Logical View instance contained in the **detail** attribute of its parent node. Other local instances which may have been retrieved locally are stored in the local cache and will be transferred according to the navigation operations performed in the Folder.

For a reference node the collection of instances exposed corresponds to the collection of Logical Views which can be referenced.

This attribute is available on root- or reference-type nodes, and on dependent nodes with a maximum cardinal value of n.

It is available for read-only access.

### Java

- Type `com.ibm.vap.generic.DataDescriptionVector` from generated `DataDescriptions`
- Internal code `rows`
- Use name `Rows`
- get/set `public DataDescriptionVector rows()`  
set not available

If the generation option “Use IBM EAB classes” is selected, another attribute is generated to simplify the use of IBM's EAB classes:

- Type `COM.ibm.ivj.javabeans.IVector`
- Internal code `iRows`
- Use name `IRows`
- get/set `public COM.ibm.ivj.javabeans.IVector iRows()`  
set not available

### COM

This information is not identified as an attribute reference in the Proxy's API. To access to this information, follow this method:

- Internal code `getRows`
- Declaration `public UcpCollection getRows()`  
Also available with a browsing API for collection-type attributes.
- Nb of elements `public Long getRowsCount()`
- Element `public {Name of generated class DataDescription} getRowsElementAt(Long i)`

## Selection of Local or Server Criterion for Instance List Sort

### Description



This attribute enables you to specify whether the collection contained in the *Instance List Presentation* attribute (page 15) is to be sorted according to the local sort criterion (**true**) or to the server sort criterion (**false**).

As a default, the instances stored in the Instance List Presentation attribute are sorted according to the local sort criterion.

This attribute is available on root-type or reference-type nodes, and on dependent nodes with a maximum cardinal value of n.

It is available for read and write access.

#### Java

- Type **Boolean**
- Internal code **localSort**
- Use name **Local Sort**
- get/set **public boolean getLocalSort() / public setLocalSort(boolean)**

#### COM

- Type **Boolean**
- Internal code **localSort**
- get/set **public Boolean isLocalSort()  
public void setLocalSort(BOOL a)**

## Local Criterion for Instance List Sort

### Description



This attribute defines the local sort criterion applied to the collection stored in the *Instance List Presentation* attribute (page 15).

As a default, the instances stored in the Instance List Presentation attribute are sorted in the order of the Logical View's key.

This attribute is available on root- or reference-type nodes, and on dependent nodes with a maximum cardinal value of n.

It is available for read and write access.

### Java

In Java, the sort criterion is represented by the instance of a class implementing the generic interface `com.ibm.vap.generic.Comparator`.

This interface is made of the following method declaration:

```
public int compare(Object a, Object b) ;
```

The implementation of this method must provide a sort criterion allowing the positioning of **a** before **b**, if `compare` returned a negative number, or **b** before **a** otherwise.

- Type
- Internal code
- Use name
- get/set

`com.ibm.vap.generic.Comparator`

`dataComparator`

-

```
public com.ibm.vap.generic.Comparator getDataComparator()  
public void setDataComparator(com.ibm.vap.generic.Comparator c)
```

### COM

Not available.

## Instance Presentation

### Description

This attribute is used to expose a particular Logical View instance (Detail). It defines all the Logical View's Data Elements which are not defined as extraction parameters.

For dependent nodes, Data Elements defining the key of parent node(s) are not exposed. Initialization of these Data Elements is automatically managed by the Folder View Proxy according to the current instances contained in the higher nodes.

When this attribute is empty, each Data Element exposed is set to an empty value, or to its default value if it is defined in the Repository.

After each direct reading or collection reading action returning only one instance, this attribute is set with the Logical View instance retrieved from the server.

This attribute is always available on root-, dependent-, and reference-type nodes.

It is available for read or write access.

#### Java

- Type `{Name of generated class DataDescription}`
- Internal code `detail`
- Use name `Detail`
- get/set `public {Name of generated class DataDescription} detail()  
/ set not available`

#### COM

- Type `{Name of generated class DataDescription}`
- Internal code `detail`
- get/set C++ `public <LPDISPATCH> getDetail () / set not available`

## Presentation of Modified Folders

### Description

This attribute exposes a list of locally modified Folders. It allows you, for example, to cancel local changes performed on a deleted Folder instance which does no longer appear in the Instance-list presentation attribute.

For each modified Folder, this attribute exposes:

- The Logical View instance of the Folder's root node.
- The number of modification services associated with the modified Folder instance.
- The modification status, which may be one of the following:
  - #Created      Locally created instance
  - #Modified     Locally modified instance
  - #Deleted      Locally deleted instance
  - #Read         Instance read, for which certain dependent instances have been updated locally.

This attribute is available on the root node of a Folder View Proxy when the options defined in the Elementary Components managing the Folder make the node modifiable.

It is available for read-only access.

### Java

- Type                    `com.ibm.vap.generic.DataDescriptionUpdateVector` from `DataDescriptionUpdate`
- Internal code         `updatedFolders`
- Use name              `Updated Folders`
- get/set                `public DataDescriptionUpdateVector updatedFolders()`  
set not available

If the generation option “`Use IBM EAB classes`” is selected, another attribute is generated to simplify the use of IBM's EAB classes:

- Type                    `COM.ibm.ivj.javabeans.IVector`
- Internal code         `iUpdatedFolders`
- Use name              `IUpdated Folders`
- get/set                `public COM.ibm.ivj.javabeans.IVector iUpdatedFolders()`  
set not available

### COM

- Nb of elements        Available with a browsing API for collection-type attributes.  
`public Long getUpdatedFoldersCount()`
- Element                `public {Name of generated class DataUpdate}`  
`getUpdatedFoldersElementAt(Long i)`

## Presentation of Modified Instances

### Description

This attribute exposes the list of the instances of the node which have been modified locally. It allows you, for example, to cancel local changes performed on a deleted instance which does no longer appear in the presentation attribute of a list of instances.

For each modified node, this attribute exposes:

- The Logical View instance of the node.
- The number of modification services associated with the modified instance.
- The modification status, which may be one of the following:
  - #Created      Locally created instance
  - #Modified     Locally modified instance
  - #Deleted      Locally deleted instance
  - #Read         Read instance, but some of its dependent instances have been updated locally.

This attribute is available on a root or dependent node of a Folder View Proxy if the Elementary Component associated with the node has an update service.

It is available for read-only access.

### Java

- Type                    `com.ibm.vap.generic.DataDescriptionUpdateVector` from `DataDescriptionUpdate`
- Internal code         `updatedInstances`
- Use name             `Updated Instances`
- get/set               `public DataDescriptionUpdateVector updatedInstances ()`  
set not available

If the generation option “`Use IBM EAB classes`” is selected, another attribute is generated to simplify the use of IBM's EAB classes:

- Type                    `COM.ibm.ivj.javabeans.IVector`
- Internal code         `iUpdatedInstances`
- Use name             `IUpdated Instances`
- get/set               `public COM.ibm.ivj.javabeans.IVector iUpdatedInstances ()`  
set not available

### COM

- Nb of elements       `public Long getUpdatedInstancesCount ()`
- Element               `public {Name of generated class DataUpdate}`  
`getUpdatedInstancesElementAt (Long i)`



## Presentation of Instances for a User Service

### Description



This attribute contains a list of Logical View instances, created by local actions during the preparation of a User Service. These instances are independent from those contained in the *Instance List Presentation* attribute (page 15).

The rules for showing the instances according to the node-hierarchy do not apply to this attribute.

These instances will be sent to the server when executing the next server-type User Service submission.

This attribute is available on root- or dependent-type nodes, when at least one User Service is defined in the Elementary Component managing the Logical View, and the latter has an iterative capacity higher than 1.

This attribute is available for read-only access.

### Java

- Type `com.ibm.vap.generic.DataDescriptionVector` from generated `UserDataDescription`
- Internal code `userInputRows`
- Use name `User Input Rows`
- get/set `public DataDescriptionVector userInputRows ()`  
set not available  
If the generation option “`Use IBM EAB classes`” is selected, another attribute is generated to simplify the use of IBM's EAB classes:
- Type `COM.ibm.ivj.javabeans.IVector`
- Internal code `iUserInputRows`
- Use name `IUser Input Rows`
- get/set `public COM.ibm.ivj.javabeans.IVector iUserInputRows ()`  
set not available

### COM

Available with a browsing API for collection-type attributes.

- Nb of elements `public Long getUserInputRowsCount ()`
- Element `public {Name of generated class UserDataDescription} getUserInputRowsElementAt (Long i)`

## Presentation of Instances Returned by a User Service

### Description



This attribute contains a list of Logical View instances, returned by the server-type User Service after its execution. These instances are independent from the instances contained in the *Instance List Presentation* attribute (page 15).

The rules for showing the instances according to the node-hierarchy do not apply to this attribute.

This attribute is available on root- or dependent-type nodes, when at least one User Service is defined in the Elementary Component managing the Logical View, and the latter has an iterative capacity higher than 1.

It is available for read and write access.

### Java

- Type `com.ibm.vap.generic.DataDescriptionVector` from generated `UserDataDescription`
  - Internal code `userOutputRows`
  - Use name `User Output Rows`
  - get/set `public DataDescriptionVector userOutputRows ()`  
set not available
- If the generation option "Use IBM EAB classes" is selected, another attribute is generated to simplify the use of IBM's EAB classes:
- Type `COM.ibm.ivj.javabeans.IVector`
  - Internal code `iUserOutputRows`
  - Use name `IUser Output Rows`
  - get/set `public COM.ibm.ivj.javabeans.IVector iUserOutputRows ()`  
set not available

### COM

- Available with a browsing API for collection-type attributes.
- Nb of elements `public Long getUserOutputRowsCount ()`
  - Element `public {Name of generated class UserDataDescription} getUserOutputRowsElementAt (Long i)`

## Presentation of an Instance Linked to a User Service

### Description

This attribute exposes a Logical View instance to be transmitted, or a Logical View instance returned by a server-type User Service. It defines all the Logical View's Data Elements which are not defined as extraction parameters.

The rules for showing the instances according to the node-hierarchy do not apply to this attribute. Consequently, in a dependent node, Data Elements defining the key of parent node(s) are exposed.

When this attribute is empty, each Data Element exposed is set to an empty value, or to its default value if it is defined in the Repository.

When a server-type User Service returns only one Logical View instance, it is exposed automatically by this attribute.

This attribute is available on root- or dependent-type nodes, when at least one User Service is defined in the Elementary Component managing the Logical View associated with the node.

It is available for read-only access.

### Java

- Type `{Name of generated class UserDataDescription}`
- Internal code `userDetail`
- Use name `User Detail`
- get/set `public {Name of generated class UserDataDescription} userDetail ()`  
set not available

### COM

- Type `{Name of generated class UserDataDescription}`
- Internal code `userDetail`
- get/set C++ `public <LPDISPATCH> getUserDetail ()` / set not available

---

## Management of User Services

### List of Available User Services

#### Description

This attribute exposes the list of User Services codes defined in the server managing the Logical View associated with the node.

This attribute is available on root- or dependent-type nodes, when at least one User Service is defined in the Elementary Component managing the Logical View associated with the node.

It is available for read-only access.

#### Java

- Type `java.lang.String[]`
- Internal code `userServiceCodes`
- Use name `User Service Codes`
- get/set `public String[] getUserServiceCodes()`  
set not available

#### COM

- Nb of elements Available with a browsing API for collection-type attributes.  
`public Long getUserServiceCodesCount()`
- Element `public String getUserServiceCodesElementAt(Long i)`

### User Service to be Executed

#### Description

This attribute defines the code of the User Service which will be processed on the server managing the node when the User Services execution is triggered on a Folder's root node.

This attribute is available on root- or dependent-type nodes, when at least one User Service is defined in the Elementary Component managing the Logical View associated with the node.

It is available for read and write access.

#### Java

- Type `java.lang.String`
- Internal code `userServiceCode`
- Use name `User Service Code`
- get/set `public String getUserServiceCode()`  
`public void setUserServiceCode(String s)`

#### COM

- Type `String`
- Internal code `userServiceCode`
- get/set C++ `public BSTR getUserServiceCode()`  
`public void setUserServiceCode(LPCTSTR s)`

---

## Management of Logical Locks

### Folder Lock Identifier

#### Description

This attribute exposes a character string calculated by the server and returned as a result of the last successful request of logical lock on a Folder instance. It is associated with the Folder instance currently contained in the root node.

This attribute is available on the root node of a Folder View Proxy when the 'logical locking' option of the Folder is set.

When the logical locking option of a Folder is set, and this attribute is empty, local updates on any of the Folder's nodes are denied.

This attribute is automatically set to an empty value for all Folder instances affected by a successful server update, or when an explicit logical unlocking action has been executed.

This attribute is available for read-only access.



No event is sent when the value of this attribute changes. Therefore, it is advised not to use attribute-to-attribute or event-to-method connections with this attribute.

#### Java

- Type `String`
- Internal code `lockTimestamp`
- Use name `Lock Timestamp`
- get/set `public String getLockTimestamp()`  
set not available

#### COM

- Type `String`
- Internal code `lockTimestamp`
- get/set C++ `public BSTR getLockTimestamp() /` set not available

---

## Management of Selection-Return Messages

### Message Label

#### Description

This attribute exposes the text of an information message returned by a server after execution of a selection action, when the end of the requested collection is reached, or when a requested instance cannot be found or is incomplete.

This attribute is always available on all types of nodes.

It is available for read-only access.

#### Java

- Type `java.lang.String`
- Internal code `accessInfoLabel`
- Use name `Access Info Label`
- get/set `public String getAccessInfoLabel ()`  
set not available

#### COM

- Type `String`
- Internal code `accessInfoLabel`
- get/set C++ `public BSTR getAccessInfoLabel ()` / set not available

### Message Key

#### Description

This attribute exposes the key of an information message returned by a server after execution of a selection action, when the end of the requested collection is reached, or when a requested instance cannot be found or is incomplete. This attribute is always available on all types of nodes.

It is available for read-only access.

#### Java

- Type `java.lang.String`
- Internal code `accessInfoKey`
- Use name `Access Info Key`
- get/set `public String getAccessInfoKey ()`  
set not available

#### COM

- Type `String`
- Internal code `accessInfoKey`
- get/set C++ `public BSTR getAccessInfoKey ()` / set not available

---

## Management of Events

### List of Events from the Last Server Action

#### Description

This attribute contains a table of integral constants representing the various events returned by the last server action.

This attribute is always available on the root node.

It is available for read-only access.

#### Java

Not available

#### COM

Available with a management API for a stack. The retrieval method for an event retrieves the first event from the stack and cancels the event from the stack.

- Nb of elements `public Long getServerEventsCount()`
- Element `public String popServerEvent()`

---

## Management of Contextual Information

### Contextual Information

#### Description

This attribute contains the Data Elements of a contextual information structure sent and received each time a server action associated with a root- or dependent-type node is executed.

This attribute is available when a User Buffer has been defined at the eBusiness Application level. It is available for read-only access.

#### Java

- Type `{Name of generated class UserContext}`
- Internal code `folderUserContext`
- Use name `Folder User Context`
- get/set `public {Name of generated class UserContext} folderUserContext()`  
set not available

#### COM

- Type `{Name of generated class UserContext}`
- Internal code `folderUserContext`
- get/set C++ `public <LPDISPATCH> getFolderUserContext() / set not available`

### Contextual Information Associated with Reference Nodes

#### Description

This attribute contains the Data Elements of a contextual information structure sent and received each time a server action associated with a reference node is executed.

It is available when a User Buffer has been defined at the level of the Elementary Component which manages the reference node.

It is available for read-only access.

#### Java

- Type `{Name of generated class UserContext}`
- Internal code `referenceUserContext`
- Use name `Reference User Context`
- get/set `public {Name of generated class UserContext} referenceUserContext() / set not available`

#### COM

- Type `{Name of generated class UserContext}`
- Internal code `referenceUserContext`
- get/set C++ `public <LPDISPATCH> getReferenceUserContext() / set not available`



---

## Available Counters

### Total Number of Local Instances

#### Description

This attribute contains the number of local instances stored in the Folder View Proxy's cache, for all nodes.

This attribute is always available on the root node.

It is available for read-only access.

#### Java

- Type `int`
- Internal code `folderInstancesCount`
- Use name `Folder Instances Count`
- get/set `public int getFolderInstancesCount()` / set not available

#### COM

This information is not referenced as an attribute in the Proxy's API. It is available through the following method:

- get/set C++ `public Long getFolderInstancesCount()` / set not available

### Total Number of Update Services

#### Description

This attribute contains the number of updates that will be performed on the various Logical View servers when the server-update execution action is next sent. This number applies to all modified Folder instances, for all nodes.

This attribute is available on the root node when at least one Logical View associated with one of the Folder's nodes is designed for update in the Elementary Component managing the Logical View.

This attribute is available for read-only access.

#### Java

- Type `int`
- Internal code `folderUpdatedInstancesCount`
- Use name `Folder Updated Instances Count`
- get/set `public int getFolderUpdatedInstancesCount()` / set not available

#### COM

This information is not referenced as an attribute in the Proxy API. It is available through the following method:

- get/set C++ `public Long getFolderUpdatedInstancesCount()`  
set not available

## Number of Update Services Associated with a Node

### Description

This attribute contains the number of updates which will be performed on the server associated with the node when the server-update execution action is next sent.

This attribute is available on each root- and dependent-type node whose associated Logical View is designed for update in the Elementary Component managing the View.

It is available for read-only access.

### Java

- Type `int`
- Internal code `nodeUpdatedInstancesCount`
- Use name `Node Updated Instances Count`
- get/set `public long getNodeUpdatedInstancesCount()`  
set not available

### COM

- This information is not referenced as an attribute in the Proxy API. It is available through the following method:
- get/set C++ `public long getNodeUpdatedInstancesCount()`  
set not available

---

## Management of Communications

### List of Available Platforms

#### Description

This attribute contains the list of logical codes (locations) of all the execution platforms available for a Folder.

When using a gateway, the first time the `getLocations` is executed for Java and `getLocationsCount` is executed for COM (either directly, or at the first server access ) a specific service call of the gateway is produced, whose role is to return the list of locations logical names, which are associated with the Folder.

This attribute is always available on the root node.

It is available for read or write access.

#### Java

When using the Java or COM version with a gateway, the first execution of `getLocations` for Java and `getLocationsCount` for COM (either directly or when first accessing the server) triggers the call of a specific service of the gateway, whose role is to return the list of location logical names associated with the Folder.

- Type `java.lang.String[]`
- Internal code `locations`
- Use name `Locations`
- get/set `public String[] getLocations()`  
set not available

#### COM

Available with a browsing API for collection-type attributes.

- Nb of elements `public Long getLocationsCount()`
- Element `public String getLocationsElementAt(Long i)`

## Platform Selected for a Query Execution

### Description

This attribute contains the logical code (location) of the next service to be executed on the server.

∞ The population existing in the local cache is not reset at a location modification. However, it is possible to cancel from the local cache all the instances of the node and its dependents with the `resetCollection` action (*Initialization of the collection* section, page 59).

This attribute is always available on the root node.

It is available for read or write access.

#### Java

- Type `String`
- Internal code `location`
- Use name `Location`
- get/set 

```
public String getLocation()
public void setLocation(String l)
```

#### COM

- Type `String`
- Internal code `location`
- get/set C++ 

```
public BSTR getLocation()
public void setLocation(LPCTSTR l)
```

## Log-in User Code

### Description

This attribute contains the user code required for logging in to the selected execution platform.

This attribute is always available on the root node.

It is available for read / write access.

#### Java

- Type `String`
- Internal code `userId`
- Use name `User Id`
- get/set 

```
get not available
public void setUserId(String u)
```

#### COM

- Type `String`
- Internal code `userId`
- get/set C++ 

```
get not available / public void setUserId(LPCTSTR u)
```

## Log-in Password

### Description

This attribute contains the password associated with the user code required for logging in to the selected execution platform.

This attribute is always available on the root node.

It is available for read / write access.

#### Java

- Type `String`
- Internal code `password`
- Use name `Password`
- get/set `get not available`  
`public void setPassword(String p)`

#### COM

- Type `String`
- Internal code `password`
- get/set C++ `get not available / public void setPassword(LPCTSTR p)`

## Name of the Machine Hosting the Java/VisualAge Pacbase Gateway

### Description

This attribute contains the TCP-IP address of the computer hosting the Communications Manager used for transmitting the messages to Elementary Components.

This attribute is always available on the root node.

When the gateway is a Java/VisualAge Pacbase gateway, this attribute MUST be set.

It is available for read or write access.

#### Java

- Type `String`
- Internal code `host`
- Use name `Host`
- get/set `public String getHost()`  
`public void setHost(String h)`

#### COM

- Type `String`
- Internal code `host`
- get/set C++ `public BSTR getHost() / public void setHost(LPCTSTR h)`

## IP Port Associated with the Communications Manager

### Description

This attribute contains the TCP-IP port associated with the Communications Manager used for transmitting the messages to Elementary Components.

This attribute is always available on the root node.

The port must be the same as that used for the gateway. As a default it is set to 5647 on both sides.

It is available for read or write access.

#### Java

- Type `int`
- Internal code `port`
- Use name `Port`
- get/set `public int getPort()`  
`public void setPort(int p)`

#### COM

- Type `Long`
- Internal code `port`
- get/set C++ `public Long getPort() / public void setPort(Long p)`

## Setting of the Platforms File's Address

### Description

This attribute contains the complete path of the platforms file used by the Communications Manager to determine the characteristics of the communication protocol providing access to an Elementary Component.

This attribute is always available on the root node.

It should be used only when the application accesses the middleware locally, not via a gateway.

It is available for read or write access.

#### Java

- get/set This information is not referenced as an attribute in the Proxy's API. It is possible to modify it by executing the following method:  
No get  
`public void setLocationsFile(String f)`

#### COM

- Type `String`
- Internal code `locationsFile`
- get/set C++ get not available / `public void setLocationsFile(LPCTSTR f)`

## Selection of the Communication Adapter

### Description

This attribute allows the definition of the communication mode used, indicating the name of the class (or dll for COM) selected **ServerAdapter** (com.ibm.vap.middleware.MiddlewareAdapter, com.ibm.vap.gateway.GatewayAdapter for Java or MwAdapter, GwAdapter for COM) or, indicating 'Direct' for a Middleware access and 'Gateway' for an access via the Gateway adapter.

For Java, it is also possible to pass directly a ServerAdapter instance.

This attribute is systematically available on the root node.

#### Java

• Type	<b>ServerAdapterName</b>
• Internal code	<b>serverAdapterName</b>
• Use name	<b>Server Adapter Name</b>
• get/set	<b>public String getServerAdapterName () public void setServerAdapterName (String className)</b>
• Type	<b>ServerAdapter</b>
• Internal code	<b>serverAdapter</b>
• Use name	<b>Server Adapter</b>
• get/set	<b>public ServerAdapter getServerAdapter () public void setServerAdapter (ServerAdapter serverAdapter)</b>

#### COM

• Type	<b>String</b>
• Internal code	<b>serverAdapterName</b>
• get/set	<b>public BSTR getServerAdapterName () public void setServerAdapterName (LPCTSTR className)</b>

## Management of Communication Parameters

### Description

The « communication parameters » attributes are read and assigned using the `getProperty/setProperty` methods. These attributes define especially the required parameters to carry out a communication with the Server Components according to the communication mode used (**Direct or Gateway**).

The following table gives the list of all parameters accepted for the above mentioned communication modes.

Note : see the end of this section for details on each parameter.

Parameters common to both communication modes	Parameters specific to the Middleware communication mode	Parameters specific to the Gateway communication mode
<code>Folder and location</code>	<code>locationsFile</code>	<code>host</code>
<code>userId</code>	<code>traceFile</code>	<code>port</code>
<code>password</code>	<code>traceLevel</code>	
<code>connectionCleaningInterval</code>	<code>nbMaxConnection</code>	
<code>hostEncoding</code>	<code>connectionTimeout</code>	
<code>clientEncoding</code>	<code>codePageFile</code>	

### Java

- Type `Object`
- Internal code `property`
- Use name `Property`
- get/set 

```
public Object getProperty (String attribut_name)/
public void setProperty(String attribut_name, Object
attribut_value)
```

### COM

- Type `Long Double String`
- Internal code `setDoubleProperty setIntProperty setStringProperty`
- Use name `setDoubleProperty setIntProperty setStringProperty`
- get/set 

```
get not available /
public void setIntProperty(LPCTSTR attribut_name, Long
value)
public void setDoubleProperty(LPCTSTR attribut_name,
double value)
public void setStringProperty(LPCTSTR attribut_name,
LPCTSTR value)
```



The following lines document the parameters accepted for the **Direct** and **Gateway** communication modes.

### **ClientEncoding**

Name or code of the character set used by the client program. If this property is set, Unicode characters to send to the server are first be converted into this "client" code page, then converted to the host code page (according to the host encoding) and finally sent.

There is no need to set the client encoding property if it is compatible with the host encoding (i.e. it contains the same characters, maybe not with the same codes).

As soon as the encoding used by the client program is NOT compatible with the host encoding (i.e. not the same character set), you should set the client encoding property to avoid characters lost.

Defaults to null, which means that characters will be directly converted from Unicode to the host code page.

### **CodePageFile**

Name of the file containing the code page conversion tables. Optional, defaults to the file CharConv.txt found in the current working directory if it exists. If no file name is given and the default file is not found, no character conversions are done.

### **ConnectionCleaningInterval**

Time in milliseconds between two cleaning of "idle" server connections. For performance purposes, the middleware layer manages a pool of underlying server connections. An idle connection is a connection in the pool that has not been used since the last cleaning.

Set this property to a small value (for example 1 second, i.e. 1000 ms), if you do not want to keep unused connections to the server (limit resource usage). Set this property to a high value (for example 60 seconds), if you want to have better performance (lower the number of connection/disconnection/re-connection).

Defaults to 60 seconds (i.e. 60000).

### **ConnectionTimeout**

Indicates the maximum time in millisecond that a thread waiting for a connection (when "nbMaxConnection" is reached) will wait before reporting a communication error to the application.

If this value is set to a small value (i.e. 1 second or less), the application will be sensible to blocking. As soon as the maximum number of connections is reached, communication errors occur.

If this value is set to a big value (i.e. infinite), the adapter never reports errors because of long waits (so, the application will not detect long waits).

Defaults to infinite.

### **location**

Name of the entry point (Location) in the locations file where to look for additional communication properties.

The location name alone determines the section to be used. The given Folder is used only when no location is given. In this case, the active location is the first one which contains a property of name FOLDER which value is equal to the given Folder. If both Folder and location are not set, the first location of the locations file is used.

**Host**

Name or IP address of the host where the VAP gateway is running. Default to 127.0.0.1, which means the local host.

**HostEncoding**

Name or code of the character encoding used by the server. Characters to send are converted into the given code page before they are sent.

The value of the host encoding property should be either an IBM code page value (ex: "37", "297") known in the active character conversion file (see the "codePageFile" property), or a code page value preceded with "Cp" (ex: "Cp37", "Cp297"), or an alias name defined in the character conversion file. If not set, the value of the property HOST\_ENCODING (or MWCODE, deprecated) found in the active location in the locations file is used. If this property is not set and no HOST\_ENCODING (and no MWCODE) property is found in the active location in the locations file, characters are not converted before they are sent.

**LocationsFile**

Name of the file used to look for additional communication properties. The content of the locations file should be organized into main sections (called Locations delimited by < >). Each Location section contains communication properties to access one specific host.

The default file name of the current working directory is "vaplocat.ini". If no valid file name is given and the default file is not found, a communication error is sent.

**NbMaxConnection**

Indicates the maximum number of server connections that can exist at the same time.

Before creating a new connection (when there is no available idle matching connection in the pool), the adapter first checks that the maximum number of connections is not reached before creating the new one. If it is reached, the last recently used idle connection is destroyed before creating the new connection. If all the connections are in use (i.e. there is no connection to destroy), the current thread is blocked until a connection is no more in use. For example, if you set this number to 1, only one connection will be created. When two threads require to communicate "at the same time" through the same connection (i.e. same connection properties), the second thread waits for its turn to use the connection.

When two threads require to communicate "at the same time" through two different connections (i.e. different connection properties), the second thread waits until the first one terminates its use of its connection, then the connection used by the first thread is destroyed before creating the connection required by the second thread. This parameter has an important impact on performance.

Defaults to infinite (no maximum, until the underlying communication API gives up).

**Password**

Password used to communicate through the middleware.  
If not set, no password is passed to the middleware layer.

**Port**

Value of the IP port on which the VAP gateway is listening for client requests.  
Default to 5647, which is the default port value used by the VAP gateway when started.

**TraceFile**

Name of the file used to write execution traces.  
The default file is an automatic created file name (with timestamp) stored in the VapTrace subdirectory of the current working directory.

**TraceLevel**

Level of detail of the execution traces :  
0 : no-traces  
1 : traces of errors only  
2 : standard non detailed traces  
3 : information traces  
4 : and upper is for debug traces.  
The default value is 1.

**UserId**

User identification used to communicate through the middleware.  
If not set, no user identification and password are passed to the middleware layer.

---

## Management of Asynchronous Conversations

### Determining the Type of Conversation

#### Description

This attribute is a Boolean value defining the current conversation type of the Folder. It must be set to **true** for recognition of an asynchronous-type conversation, to **false** for a synchronous-type conversation. As a default, it is set to **false**.

This attribute is always available on the root node.

It is available for read / write access.

#### Java

- Type **Boolean**
- Internal code **asynchronous**
- Use name **Asynchronous Mode**
- get/set **public Boolean isAsynchronous()  
public void setAsynchronous(Boolean isAsynchronous)**

#### COM

- Type **Boolean**
- Internal code **asynchronous**
- get/set C++ **public BOOL getAsynchronous()  
public void setAsynchronous(BOOL a)**

### Last Identifier of an Asynchronous Conversation

#### Description

This attribute contains the identifier of the reply for last query performed with an asynchronous-type conversation on the current location.

This attribute is always available on the root node.

It is available for read access.

#### Java

- Type **com.ibm.vap.generic.ServerActionContext**
- Internal code **lastReplyContext**
- Use name **Last Reply Context**
- get/set **public ServerActionContext getLastReplyContext()  
set not available**

#### COM

- Type **ServerActionContext**
- Internal code **lastReplyContext**
- Use name **lastReplyContext**
- get/set C++ **public LPDISPATCH getLastReplyContext()  
set not available**

## Maximum Number of Pending Replies

### Description

This attribute contains the maximum number of reply-pending queries for the current location. This number is a specific parameter (**MWMAXREPLY**) of the asynchronous conversations, specified in the platforms file.

This attribute is always available on the root node.

It is available for read / write access.

#### Java

- Type `int`
- Internal code `maximumReplyCount`
- Use name `Maximum Reply Count`
- get/set `public int getMaximumReplyCount()`  
`public void setMaximumReplyCount(int maximumReplyCount)`

#### COM

- Type `int`
- Internal code `maximumReplyCount`
- get/set C++ `public short getMaximumReplyCount()` / set not available

## Number of Pending Replies

### Description

This attribute contains the number of asynchronous replies pending for a Folder. It is set to zero after each location change.

It is incremented when executing any query using an asynchronous-type conversation, except for update-type queries.

It is decremented after each return of a reply, or when pending queries are canceled.

This attribute is always available on the root node.

It is available for read / write access.

#### Java

- Type `int`
- Internal code `pendingReplyCount`
- Use name `Pending Reply Count`
- get/set `public int getPendingReplyCount()`  
set not available

#### COM

- Type `int`
- Internal code `pendingReplyCount`
- get/set C++ `public short getPendingReplyCount()`  
set not available

---

## Conversation Time

### Communication Time

#### Description

This attribute contains the total communication time of the last conversation with the server.

It is set to zero.

This attribute is always available on the root node.

It is available for read access.

#### Java

	Time, stated in milliseconds.
• Type	<code>int</code>
• Internal code	<code>communicationResponseTime</code>
• Use name	<code>Communication Response Time</code>
• get/set	<code>public int getCommunicationResponseTime ()</code> set not available

#### COM

• Type	<code>long</code>
• Internal code	<code>communicationResponseTime</code>
• get/set C++	<code>public long getCommunicationResponseTime ()</code> set not available

### Execution Time of the Server Processing

#### Description

This attribute contains the total execution time of the server processing for the last conversation.

It is set to zero.

This attribute is always available on the root node.

It is available for read access.

#### Java

	Time, stated in milliseconds.
• Type	<code>int</code>
• Internal code	<code>serverResponseTime</code>
• Use name	<code>Server Response Time</code>
• get/set	<code>public int getServerResponseTime ()</code> set not available

#### COM

• Type	<code>long</code>
• Internal code	<code>serverResponseTime</code>
• get/set C++	<code>public long getServerResponseTime ()</code> / set not available

---

## Sub-Schema Management

### List of Available Sub-Schemas

#### Description

This attribute exposes the list of the sub-schemas available on the node. Sub-schemas are specified in the description of the Logical View associated with the node.

This attribute is available if the Elementary Components manage the presence of Data Elements (options **VECTPRES=YES** or **CHECKSER=YES**) and if the node includes at least one sub-schema.

It is available for read-only access.

#### Java

- Type **SubSchema []**
- Internal code **subSchemaList**
- Use name **SubSchema List**
- get/set **public SubSchema [] getSubSchemaList()**  
set not available

#### COM

- This action is not referenced as an attribute in the Proxy's API. It is possible to modify it by executing the following method:
- Internal code **getSubSchemas**
  - Declaration **public VapCollection getSubSchemas()**  
Also available with a browsing API for collection-type attributes.
  - Nb of elements **public long getSubSchemasCount()**
  - Element **public LPDISPATCH getSubSchemasElementAt(long i)**

## Sub-schema to be Taken into Account

### Description

This attribute contains the sub-schema to be taken into account when a selection, read or update action is performed.

Sub-schemas are specified in the description of the Logical View associated with the node.

This attribute is available if the Elementary Components manage the presence of Data Elements (options **VECTPRES=YES** or **CHECKSER=YES**) and if the node includes at least one sub-schema.

It is available for read and write access.

#### Java

- Type **SubSchema**
- Internal code **subSchema**
- Use name **Current SubSchema**
- get/set **public String getSubSchema()  
public void setSubSchema(SubSchema SubSchema)**

#### COM

- Type **VapSubSchema**
- Internal code **subSchema**
- get/set C++ **LPDISPATCH getSubSchema() / void setSubSchema(LPDISPATCH s)**

---

## External Request Management

### Access and Set a Request

#### Operation

This attribute returns and sets the current request for the Proxy.

The set request method enables to have the Proxy participate in the storage context of actions started by an other Proxy instance.

#### Java

- Type **MainRequest**
- Internal code **request**
- User name **MainRequest**
- get/set **public MainRequest getRequest() /  
public void setRequest(MainRequest request)**

#### COM

- Type **VapRequest**
- Internal code **Request**
- get/set C++ **LPDISPATCH getRequest() / void setRequest(LPDISPATCH)**



---

## Use of a JTable

### Display of the Instances Collection in a JTable

#### Description

This attribute is available with Java only.

It enables you to insert a **JTable**, which is a swing component constituted of several rows and columns, and to display the collection of Logical View instances in this **JTable** via the **tableModel** attribute.

This attribute is available on all node types if you selected the generation option **Use Swing**.

It is available for read and write access.

#### Java

- Type `PacbaseTableModel`
- Internal code `tableModel`
- Use name `TableModel`
- get/set `public getTableModel`  
`public setTableModel(PacbaseTableModel)`

#### COM

Not available

### Display of the Updated Folders in a JTable

#### Description

This attribute is available with Java only.

It enables you to insert a **JTable**, which is a swing component constituted of several rows and columns, and to display the collection of updated Folders in this **JTable** via the **updatedFoldersTableModel** attribute.

This attribute is available on all root-type nodes if you selected the generation option **Use Swing**.

It is available for read and write access.

#### Java

- Type `PacbaseUpdateTableModel`
- Internal code `updatedFoldersTableModel`
- Use name `UpdatedFoldersTableModel`
- get/set `public getUpdatedFoldersTableModel`  
`public`  
`setUpdatedFoldersTableModel(PacbaseUpdateTableModel)`

#### COM

Not available

## Display of the Updated Instances in a JTable

### Description

This attribute is available with Java only.

It enables you to insert a **JTable**, which is a swing component constituted of several rows and columns, and to display the collection of updated instances in this **JTable** via the **updatedInstancesTableModel** attribute.

This attribute is available on all root-type nodes if you selected the generation option **Use Swing**.

It is available for read and write access.

#### Java

- Type **PacbaseUpdateTableModel**
- Internal code **updatedInstancesTableModel**
- Use name **UpdatedInstancesTableModel**
- get/set **public getUpdatedInstancesTableModel**  
**public**  
**setUpdatedInstancesTableModel (PacbaseUpdateTableModel)**

#### COM

Not available

## Display of the Instance Collection in input/output by a User Service in a JTable

### Description

This attribute is available with Java only.

It enables you to insert a **JTable** in an application, which is a swing component constituted of several rows and columns and to display the collection of Logical View instances in input/output by a User Service in this **JTable** via the **tableModel** attribute.

This attribute is available on all node types if you have selected the generation option **Use Swing**.

It is available for read and write access.

#### Java

- Type **PacbaseTableModel**
- Internal code **tableModel**
- Use name **TableModel**
- get/set **public getUserInputTableModel**  
**public setUserInputTableModel (PacbaseTableModel)**  
**public getUserOutputTableModel**  
**public setUserOutputTableModel (PacbaseTableModel)**

#### COM

Not available

---

## Chapter 3. Actions

An action is a piece of processing which can be executed by the Proxy. When an action requires some parameters for its execution, or when it returns results, those are passed through by the Logical Views attributes.

There are two types of actions for a Proxy:

- **Local actions**, which perform update operations on Logical View instances memorized by the Proxy.
- **Server actions**, which perform specific processing on the server. If the server uses a User Buffer, this type of action exchanges its contents every time it holds a conversation with the server.

Actions can therefore trigger either local processing internal, to the Logical View, or remote processing. These are standard selections, update processing actions, and user processing actions defined for the Elementary Components associated with the Logical Views.

**Note:** The availability of these actions is indicated in the “Operation” paragraph for each action. In the case of a Java target, if an action is used although it is not available (wrong usage of the public method), a `java.lang.IllegalStateException` exception will be raised.

---

### Actions Performed Locally

#### Updates

##### Creation of a Logical View Instance

###### Operation

This action creates a Logical View instance locally.

This action is valid if:

- The instance does not exist locally.
- Checks performed on all the instance's Data Elements did not return any errors.
- The parent instance of a dependent node is present locally.
- For a dependent node with a maximum cardinal value of 1, the created instance is the only one present locally for the parent instance (i.e., the parent instance has no dependent instance so far).
- The Folder has “modifiable” status.

If the action is valid:

- The “Total number of update services” counter is incremented by 1.
- The “Number of update services” counter associated with the node is incremented by 1.
- The “Total number of local instances” counter is incremented by 1.
- The new instance is included in the instance-list container associated with the node.

- The new modification is included in the modified Folders' presentation attributes.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is available if the Elementary Component allows for updates on the Logical View, and if all dependent nodes with a minimum cardinal value of 1 are present in the Folder View.

#### Java

- Declaration `public void createInstance() throws LocalException`
- Use name `Create Instance`

#### COM

- C++ declaration `public void createInstance()`
- Internal code `createInstance()`

### Modification of a Logical View Instance

#### Operation

This action modifies a Logical View instance locally.

This action is valid if:

- The instance exists in the Instance-presentation attribute.
- The instance exists locally.
- Checks performed on all the instance's Data Elements did not return any errors.
- The Folder has "modifiable" status.

If the action is valid:

- The "Total number of update services" counter is incremented by 1 if no update transaction is currently associated with this instance.
- The "Number of update services" counter associated with the node is incremented by 1 if no update transaction is currently associated with this instance.
- The modification is included in the instance-list container associated with the node.
- The new modification is included in the modified Folders' presentation attributes.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is available if the Elementary Component allows for updates on the Logical View.

## Java

- Declaration `public void modifyInstance() throws LocalException`
- Use name `Modify Instance`

## COM

- C++ declaration `public void modifyInstance()`
- Internal code `modifyInstance()`

## Deletion of a Logical View Instance

### Operation

This action deletes a Logical View instance locally. It locally deletes all dependent nodes' instances one after the other.

This action is valid if:

- The instance exists locally.
- The instance exists in the Instance-presentation attribute.
- The parent instance of a dependent node is present locally.
- The Folder has “modifiable” status.

If the action is valid:

- The “Total number of update services” counter is incremented by 1 if no update transaction is currently associated with this instance.
- The “Number of update services” counter associated with the node is incremented by 1 if no update transaction is currently associated with this instance.
- The “Total number of local instances” counter is decremented by the number of dependent instances implicitly deleted, + 1.
- The instance is deleted from the instance-list container associated with the node.
- The new modification is included in the modified Folders' presentation attributes.
- All local instances which depend on the deleted instance are deleted.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is available if the Elementary Component allows for updates on the Logical View.

## Java

- Declaration `public void deleteInstance() throws LocalException`
- Use name `Delete Instance`

## COM

- C++ declaration `public void deleteInstance()`
- Internal code `deleteInstance()`

## Cancellation of Updates

### Cancellation of a Folder's Updates

#### Operation

This action cancels all local updates performed on a Folder's instance, for all nodes, starting with the first local update.

This action is valid if:

- The instance exists in the root node's Instance-presentation attribute.

If the action is valid:

- The initial image of the instance and dependent instances is restored in the local cache, in the presentation attributes and in the instance list containers.
- The update-services total number counter is recalculated.
- The update-services number counter associated with the node is recalculated.
- The "Total number of local instances" counter is recalculated.
- The instance is deleted from the modified Folders' presentation attributes.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is available if at least one of the Elementary Components of the Folder allows for updates on a Logical View it manages.

#### Java

- Declaration `public void undoLocalFolderUpdates({Name of generated class DataUpdate} d) throws LocalException`
- Use name `Undo Local Folder Updates`

#### COM

- C++ declaration `public void undoLocalFolderUpdates(LPDISPATCH d)`
- Internal code `undoLocalFolderUpdates({Name of generated class DataUpdate})`

## Cancellation of all Folders Updates

### Operation

This action cancels all the local updates performed on all Folder instances, for all nodes.

Action impact:

- The initial images of the Folder instances and all their dependent instances are restored in the local cache, in the presentation attributes and in the instance-list containers.
- The “Total number of update services” counter is reset.
- The “Number of update services” counters associated with each node are reset.
- The “Total number of local instances” counter is recalculated.
- The modified Folders' presentation attributes are reset.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is available if at least one of the Elementary Components of the Folder allows for updates on a Logical View it manages.

### Java

- Declaration `public void undoAllLocalFolderUpdates ()`
- Use name `Undo All Local Folder Updates`

### COM

- C++ declaration `public void undoAllLocalFolderUpdates ()`
- Internal code `undoAllLocalFolderUpdates ()`

## Cancellation of Updates on a Node Instance

### Operation

This action cancels all local updates performed on an instance of the node, starting with the first local update. This action takes a node instance as a parameter.

This action is valid if:

- The instance passed as a parameter is a node instance which has been locally updated.

If the action is valid:

- The initial image of the instance and dependent instances (if the instance has a #Deleted or #Created status) is restored in the local cache and in the presentation instances attributes.
- The counter of the total number of update services is recalculated.
- The counter of the update services number associated with the node is recalculated.
- The counter of the total number of local instances is recalculated.
- The updated instance and all its dependent instances are deleted from the presentation attributes of modified instances.
- The Folders' presentation attribute is updated.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is available on a root or dependent node of a Folder View Proxy if the Elementary Component associated with the node includes an update service.

### Java

- Declaration `public void undoLocalUpdate({Name of generated class  
DataDescriptionUpdate} d) throws LocalException`
- Use name `Undo Local Update`

### COM

- C++ declaration `public void undoLocalUpdate(LPDISPATCH d)`
- Internal code `undoLocalUpdate({Name of generated class  
DataDescriptionUpdate} d)`



## Cancellation of Updates on all the Instances of a Node

### Operation

This action cancels all the local updates performed on the instances of a node and of its current hierarchy, since the first local update.

Action impact:

- The initial images of the instances of the node for the current hierarchy and of all their dependent instances (if the modification status of a node instance is not #Modified) are restored in the local cache, in the presentation attributes and in the instance-list containers.
- The "total number of update services" counter is recalculated.
- The "number of update services" counter of the number of update services associated with the node is reset.
- The "total number of local instances" counter is recalculated.
- The updated instances and all their dependent instances are deleted from the presentation attribute of modified instances.
- The modified Folders' presentation attribute is updated.
- The no-error-detection event is sent.

This action is available on a root or dependent node of a Folder View Proxy if the Elementary Component associated with the node includes an update service.

#### Java

- Declaration `public void undoAllLocalUpdate ()`
- Use name `Undo All Local Update`

#### COM

- C++ declaration `public void undoAllLocalUpdate ()`
- Internal code `undoAllLocalUpdate ()`

## Management of User Services

### Assignment of an Instance to a User Service

#### Operation

On a node, this action locally creates a new Logical View instance, reserved for the execution of the next User Service.

This action is valid if:

- The instance exists in the Instance-presentation attribute of an instance linked to a User Service.

If the action is valid:

- The counter of Logical View instances reserved for a User Service is incremented by 1.
- The instance is included in the presentation attributes of instances reserved for a User Service.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is available when the Elementary Component associated with the node manages at least one User Service.

#### Java

- Declaration `public void createUserInstance() throws LocalException`
- Use name `Create User Instance`

#### COM

- C++ declaration `public void createUserInstance ()`
- Internal code `createUserInstance ()`

## Modification of an Assigned Instance

### Operation

On a node, this action locally modifies a Logical View instance reserved for the execution of the next User Service.

This action is valid if:

- The instance exists in the presentation attribute of instances reserved for a User Service.
- The instance exists in the Instance-presentation attribute of an instance linked to a User Service.

If the action is valid:

- The modification is included in the presentation attribute of instances designed for a User Service.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is available when the Elementary Component associated with the node manages at least one User Service.

#### Java

- Declaration `public void modifyUserInstance() throws LocalException`
- Use name `Modify User Instance`

#### COM

- C++ declaration `public void modifyUserInstance()`
- Use name `modifyUserInstance()`

## Deletion of an Assigned Instance

### Operation

On a node, this action locally deletes a Logical View instance reserved for the execution of the next User Service.

This action is valid if:

- The instance exists in the presentation attribute of instances reserved for a User Service.
- The instance exists in the presentation attribute of an instance linked to a User Service.

If the action is valid:

- The counter of Logical View instances reserved for a User Service is decremented by 1.
- The presentation attribute of instances designed for a User Service integrates the instance deletion.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is available when the Elementary Component associated with the node manages at least one User Service.

#### Java

- Declaration `public void deleteUserInstance() throws LocalException`
- Use name `Delete User Instance`

#### COM

- C++ declaration `public void deleteUserInstance()`
- Internal code `deleteUserInstance()`

## Local Navigation in the Folders

### Global Instance-presentation attributes setting rule:

When the **detail** attribute (instance-presentation) of a parent node contains a valid instance, the **detail** and **rows** attributes (presentation of the instances list) of its dependent nodes are set according to the following rules:

- If the dependent node has a maximum cardinal values of n, its **rows** attribute is set with all the instances stored in the local cache and depending on the current instance of the parent node. If there is only one instance in the local cache, its **detail** attribute is also set with this instance.
- If the node has a maximum cardinal value of 1, its **detail** attribute is set with the instance depending on the parent node's current instance, if it is found in the local cache.
- If the node does not meet any of the above rules, its Instances-presentation attributes are empty.

### Current Selection of an Instance in a Folder

#### Operation

This action assigns to the **detail** attribute of a node an instance of the same type, such as, in particular, an instance from the **rows** attribute.

This action is valid if:

- The input parameter for this action is an instance from **DataDescription**.

If the action is valid:

- The **detail** attribute contains the instance to be assigned.
- The **detail** and **rows** attributes of the dependent nodes are set according to the Global setting rule.
- The Folder's lock identifier is set if the current instance belongs to the root node which is currently locked.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is always available.

#### Java

- Declaration `public void getDetailFromDataDescription({Name of generated class DataDescription} d) throws LocalException`
- Use name `Get Detail From Data Description`

#### COM

- C++ declaration `public void getDetailFromData(LPDISPATCH d)`
- Internal code `getDetailFromData({Name of generated class DataDescription})`

## Selection of an instance from an Index

### Operation

This method enables to reactivate the current selection according to the index of a Logical View instance contained in the "rows" collection.

#### Java

- Declaration `public void getDetailFromRowIndex(int index) throws LocalException`
- Use name `Get Detail From Rows Index`

#### COM

- C++ declaration `public void getDetailFromRowIndex (short index)`
- Internal code `getDetailFromRowIndex(Integer)`

## Selection of an Instance Associated with a User Service

### Operation

This action assigns to the **UserDetail (Presentation of an instance for a User Service)** attribute of a node an instance of the same type, such as, in particular, an instance from the **UserRows (Presentation of an instance-list for a User Service)** attribute.

Action impact:

- The presentation attribute of an instance reserved for a User Service contains the instance to be assigned.
- The no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is available when the Elementary Component associated with the node has at least one User Service.

#### Java

- Declaration `public void getUserDetailFromDataDescription({Name of generated class DataDescription} d) throws LocalException`
- Use name `Get User Detail From Data Description`

#### COM

- C++ declaration `public void getUserDetailFromData (LPDISPATCH d)`
- Internal code `getUserDetailFromData({Name of generated class DataDescription} d)`

## Reactivation of the Current Selection

### Operation

This action enables to reactivate the current selection according to a Logical View instance.

Note: The Logical View instance is not always retrieved from the **rows** collection. It may have been created only to meet the Development needs.

Example : After the selection of 300 "Client" Logical View instances, creation of one "Client" Logical View instance which is assigned the client number 56 and use of the "restoreSelection" method. The current selection of the Proxy is fed with the client 56 first retrieved and the hierarchy of the dependants is activated with client number 56 as the root.

#### Java

- Declaration `public void restoreSelection ({Name of generated class }Data d)`
- Use name `restoreSelectionFromData ({Name of generated class}Data d)`

#### COM

Not available

## Miscellaneous Initializations

### Initialization of the collection

#### Operation

This action enables to discard the node instances and its dependent nodes from the local cache.

Action impact:

- the **detail** attribute of the node is re-initialized.
- the **rows** attribute of the node is re-initialized.
- the **detail** attributes of dependents nodes are re-initialized.
- the **rows** attributes of dependents nodes are re-initialized.

The action is always available for all nodes.

#### Java

- Declaration `public void resetCollection()`
- Use name `Reset Collection`

#### COM

- C++ declaration `public void resetCollection()`
- Internal code `resetCollection()`

## Initialization of Extraction Methods

### Operation

This action sets the **Extraction method to be executed** attributes of the node and all its dependent nodes to empty values.

The “Extraction method to be executed” attribute of each affected node contains an empty value.

This action is available for all nodes on which at least one extraction method has been defined in the Elementary Component.

#### Java

- Declaration `public void resetExtractMethodCodes ()`
- Use name `Reset Extract Method Codes`

#### COM

- C++ declaration `public void resetExtractMethodCodes ()`
- Internal code `resetExtractMethodCodes ()`

## Initialization of the User Services

### Operation

This action sets the **User Service to be executed** attributes of the node and all its dependent nodes to empty values.

The “User Service to be executed” attribute of each affected node contains an empty value.

This action is available when the Elementary Component associated with the node has at least one User Service.

#### Java

- Declaration `public void resetUserServiceCodes ()`
- Use name `Reset User Service Codes`

#### COM

- C++ declaration `public void resetUserServiceCodes ()`
- Internal code `resetUserServiceCodes ()`



## Initialization of the “Presentation of Instances for a User Service” Container

### Operation

This action sets the **Presentation of instances for a User Service** attributes of the node and all its dependent nodes to empty values.

Action impact:

- For each affected node, the “Presentation of instances designed for a User Service to be executed” attribute contains an empty value.

This action is available when the Elementary Component associated with the node has at least one User Service.

#### Java

- Declaration `public void resetUserRows ()`
- Use name `Reset User Rows`

#### COM

- C++ declaration `public void resetUserRows ()`
- Internal code `resetUserRows ()`

## Initialization of the Update-Refresh Option

### Operation

This action inhibits the update-refresh option on the node and on its dependent nodes, by setting the corresponding Boolean value to “false”.

This action is available when the Elementary Component associated with the node allows for updates on the Logical View it manages.

#### Java

- Declaration `public void resetAllRefreshOption ()`
- Use name `Reset All Refresh Option`

#### COM

- C++ declaration `public void resetAllRefreshOption ()`
- Internal code `resetAllRefreshOption ()`

## Initialization of Selection Criteria

### Operation

This action sets the Selection Criteria attributes of the node and all its dependent nodes to empty values.

As a result of this action, the Selection Criteria attribute of each affected node contains an empty value.

This action is always available for all root- and dependent-type nodes.

#### Java

- Declaration `public void resetSelectionCriteria()`
- Use name `Reset Selection Criteria`

#### COM

- C++ declaration `public void resetSelectionCriteria()`
- Internal code `resetSelectionCriteria()`

## Addition of instances in the local cache without server access

### Operation

This action allows the addition, in the local cache, of non-read instances from the server. These instances have not the locally-created status.

This action is valid if the instance does not exist in local mode, whatever its status is.

If the action is valid:

- The "Total number of local instances" counter is incremented by 1.
- The instance list container associated with the node embeds the new instance
- The event 'no error detected' is sent.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is always available on all nodes.

#### Java

- Declaration `public void initializeInstance() throws LocalException`
- Use name `Initialize Instance`

#### COM

- C++ declaration `public void initializeInstance()`
- Internal code `initializeInstance()`

## Management of Referenced Instances

### Assignment of a Referenced Instance

#### Operation

This action maps the identifier-type Data Elements in the reference-node instance used as a parameter of the action to the 'foreign key'-type Data Elements of the referencing-node instance.

This action is valid if:

- An instance exists in the Instance-presentation attribute of the referencing node.
- The reference-node's instance does not contain an empty value.

If the action is valid:

- The 'foreign key'-type Data Elements in the Instance-presentation attribute of the referencing node are initialized with the identifier-type Data Elements of the reference node.

If the action is not valid:

- The error is added to the Error Object.
- A local error event is sent.

This action is always available on reference nodes.

#### Java

- Declaration `public void transferReferenceFromSelectedRow({Name of generated class DataDescription} d) throws LocalException`
- Use name `Transfer Reference From Selected Row`

#### COM

- C++ declaration `public void transferReferenceFromSelectedRow(LPDISPATCH d)`
- Internal code `transferReferenceFromSelectedRow({Name of generated class DataDescription} d)`

## Retrieval of Proxies' Generation Contexts

### Generation Context of a Folder

#### Operation

This action retrieves the VisualAge Pacbase constants from the Services Manager associated with the root node, in the form of a collection of character strings containing the following information:

- ◆ Services Manager external name
- ◆ VisualAge Pacbase code of the Folder (or Elementary Component)
- ◆ Database code of the VisualAge Pacbase Repository
- ◆ Library code
- ◆ Generation-session number
- ◆ User code
- ◆ Generation date
- ◆ Generation time
- ◆ Folder View code

This action is always available for a root node.

#### Java

- Declaration `public String[] getFolderConstants()`
- Use name `Get Folder Constants`

#### COM

Available with a browsing API for collection-type attributes.

- Nb of elements `public Long getFolderConstantsCount()`
- Element `public String getFolderConstantsElementAt(Long i)`

## Generation Context of a Node

### Operation

This action retrieves the VisualAge Pacbase constants from the Elementary Component associated with the node.

Action impact:

- The action returns a collection of character strings containing the following information:
  - ♦ External name of the Elementary Component
  - ♦ VisualAge Pacbase code of the Elementary Component
  - ♦ Database code of the VisualAge Pacbase Repository
  - ♦ Library code
  - ♦ Generation-session code
  - ♦ User code
  - ♦ Generation date
  - ♦ Generation time
  - ♦ Operations version of the Elementary Component

This action is always available for all types of nodes (root, dependent, or reference).

#### Java

- Declaration `public String[] getNodeConstants()`
- Use name `Get Node Constants`

#### COM

- Available with a browsing API for collection-type attributes.
- Nb of elements `public Long getNodeConstantsCount()`
- Element `public String getNodeConstantsElementAt(Long i)`

## Sub-Schema Management

### No Selection of Sub-Schema

#### Operation

This action resets the `subSchema` attribute, that is no sub-schema is selected.

This action is available if the Elementary Components manage the presence of Data Elements (`VECTPRES=YES` or `CHECKSER=YES`) and if the node includes at least one sub-schema.

#### Java

- Declaration `public void resetSubSchema()`
- Use name `Reset SubSchema`

#### COM

- C++ declaration `public resetSubSchema()`
- Internal code `resetSubSchema()`

---

## Actions Performed on a Remote Server

Reminder: Global Instance-presentation attributes setting rule:

When the Instance-presentation attribute of a parent node contains a valid instance, the Instance- and Instance-list-presentation attributes of the dependent nodes are set according to the following rules:

- If the dependent node has a maximum cardinal value of  $n$ , its Instance-list-presentation attribute is set with all the instances stored in the local cache and dependent on the current instance of the parent node. If there is only one instance in the local cache, the Instance-presentation attribute is also set with this instance.
- If the node has a maximum cardinal value of 1, its Instance-presentation attribute is set with the instance dependent on the parent node's current instance, if it is found in the local cache.
- If the node does not meet the above-stated rules, its Instance-presentation and Instance-list-presentation attributes are set to empty values.

### Selection on a Node

#### Selection of a Set of Instances

##### Operation

This action defines a Logical View instance collection associated with the node, and retrieves all of this collection's instances, or the first page.

If the action is valid:

- The Instance-list-presentation attribute is modified according to the value of the management mode attribute of the collection.
- The "Total number of local instances" counter is initialized.
- The selection-return message's label and key are initialized if the last instance of the collection has been retrieved.
- The Instance- and Instance-list presentation attributes of dependent-type nodes are set to empty values.
- A no-error-detection event is sent.
- The event "Presence of updatable local instances" is sent, with a collection management in automatic mode, if updatable instances are still present in the local cache.
- The event "Retrieval of a collection's first page" is sent if the Folder operates in **non extend** mode and with the collection management in automatic mode.
- The event "Presence of at least one following page" is sent if the collection's last instance has not been retrieved.
- The event "Retrieval of the last page" is sent if the last collection's instance has been retrieved.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes –if they are present– are set.

This action is always available on root- and reference-type nodes.

#### Java

- Declaration `public void selectInstances() throws ServerException, CommunicationError, SystemError, LocalException`
- Use name `Select Instances`

#### COM

- C++ declaration `public void selectInstances()`
- Internal code `selectInstances()`

### Reading of an Instance with or without Logical Locking

#### Operation

This action retrieves a Logical View instance associated with the node, and appropriates it for exclusive update, if relevant.

This action is valid if:

- The instance is not already locked, in the case of an action with locking.

If the action is valid:

- The Instance-presentation attribute is set.
- The Instance-list-presentation attribute is modified according to the value of the management mode attribute of the collection.
- The “Total number of local instances” counter is initialized.
- The selection-return message's label and key are initialized if the last instance of the collection has not been retrieved.
- The Instance- and Instances-lists- presentation attributes of dependent nodes are set to empty values.
- A no-error-detection event is sent.
- The event “Presence of updatable local instances” is sent, with the collection management in automatic mode, if updatable instances are still present in the local cache.
- The Folder-lock identifier is initialized in the case of a locking request.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.
- The Folder-lock identifier is set to an empty value in the case of a locking request, and the Folder changes to the ‘not-modifiable’ status.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes –if they are present– are set.

This action is always available on all nodes.

## Java

- Declaration `public void readInstance() throws LocalException, ServerException, CommunicationError, SystemError`  
`public void readInstanceAndLock() throws LocalException, ServerException, CommunicationError, SystemError`
- Use name `Read Instance`  
`Read Instance And Lock`

## COM

- C++ declaration `public void readInstance()`  
`public void readInstanceAndLock()`
- Use name `readInstance()`  
`readInstanceAndLock()`

### Reading of Instances from identifiers

This action defines a Logical View instance collection associated with the node, and retrieves the instances whose keys passed as a parameter.

The keys collection passed as a parameter can include `SelectionCriteria` class instances or `DataDescription` classes.

If this action is valid:

- the `rows` attribute is entered according to the management mode of the collection.
- The “Total number of local instances” counter of the Folder is initialized.
- The selection-return message's label and key are initialized if the last instance has been retrieved.
- The `detail` and `rows` attributes of the dependent nodes are set to empty values.
- A no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes –if they are present– are set.

This action is always available on all root- and dependent-type nodes.

## Java

- Signature `public void readInstances(Enumeration keys) throws ServerException, LocalException, CommunicationError, SystemError`
- User name `Read Instances`

## COM

- C++ declaration `public void readInstances(LPDISPATCH keys)`
- Internal code `readInstances(VapCollection)`



## Concurrent Selection on Multiple Nodes with or without Locking

### Reading of an Instance and its Immediate Hierarchy

#### Operation

This action retrieves a Logical View instance associated with the node, and appropriates it for exclusive update –if relevant, then retrieves all or part of the instances of first-level dependent nodes.

This action is valid if:

- The instance is not locked, in the case of an action with locking.

If the action is valid:

- The Instance-presentation attribute is initialized with the result of the selection.
- The Instance-list-presentation attribute is modified according to the value of the management mode attribute of the collection.
- The “Total number of local instances” counter is initialized.
- The selection-return message's label and key are initialized for each first-level dependent node if the collection's last instance has been retrieved.
- The Instance- and Instance-list-presentation attributes of first-level dependent nodes in the hierarchy are initialized with the result of the selection, except for those for which the number of exchanged instances was set to zero (they are initialized to empty values). For the Instance-list attribute, the modification is made according to the value of the collection management mode attribute, the Instance-list attribute being associated with each node.
- The Instance- and Instance-list-presentation attributes of dependent nodes of a hierarchical level higher than one are set to empty values.
- A no-error-detection event is sent.
- The event “Presence of updatable local instances” is sent, with the collection management in automatic mode, if updatable instances are still present in the local cache.
- A “Record not found” event is sent on every node participating in the selection and having a maximum cardinal value of 1, and whose instance has not been retrieved.
- The Folder-lock identifier is set to an empty value in the case of a locking request.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.
- The Folder-lock identifier is set to an empty value in the case of a locking request, and the Folder changes to the ‘not-modifiable’ status.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes –if they are present– are set.

This action is always available on all root- and dependent-type nodes.

## Java

- Declaration 

```
public void readInstanceWithFirstChildren() throws  
LocalException, ServerException, CommunicationError,  
SystemError  
public void readInstanceWithFirstChildrenAndLock() throws  
LocalException, ServerException, CommunicationError,  
SystemError
```
- Use name 

```
Read Instance With First Children  
Read Instance With First Children And Lock
```

## COM

- C++ declaration 

```
public void readWithFirstChildren()  
public void readWithFirstChildrenAndLock()
```
- Internal code 

```
readWithFirstChildren()  
readWithFirstChildrenAndLock()
```

## Reading of an Instance and its Complete Hierarchy

### Operation

This action retrieves a Logical View instance associated with the root node, appropriates it for exclusive update if relevant, and retrieves all the instances of each dependent node, whatever its hierarchical level.

This action is valid if:

- The instance is not already locked, in the case of a action with locking.

If the action is valid:

- The Instance-presentation attribute of the root node is initialized with the selection result.
- The Instance-list-presentation attribute of the root node is modified according to the value of the collection management mode attribute.
- The “Total number of local instances” counter is initialized.
- The selection-return message’s label and key are initialized for each dependent node if the collection’s last instance has been retrieved.
- The Instance- and Instance-list-presentation attributes of a dependent node are set according to the Global setting rule. For the Instance-list attribute, the modification is made according to the value of the collection management mode attribute, the Instance-list attribute being associated with each node.
- A no-error-detection event is sent.
- The event “Presence of updatable local instances” is sent, with the collection management in automatic mode, if updatable instances are still present in the local cache.
- A “Record not found” event is sent on every node participating in the selection and having a maximum cardinal value of 1, and whose instance has not been retrieved.
- The Folder-lock identifier is set to an empty value in the case of a locking request.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.
- The Folder-lock identifier is set to an empty value in the case of a locking request, and the Folder changes to the 'not-modifiable' status.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes –if they are present– are set.

This action is always available on root nodes.

#### Java

- Declaration 

```
public void readInstanceWithAllChildren() throws
LocalException, ServerException, CommunicationError,
SystemError
public void readInstanceWithAllChildrenAndLock() throws
LocalException, ServerException, CommunicationError,
SystemError
```
- Use name 

```
Read Instance With All Children
Read Instance With All Children And Lock
```

#### COM

- C++ declaration 

```
public void readWithAllChildren()
public void readWithAllChildrenAndLock()
```
- Internal code 

```
readWithAllChildren()
readWithAllChildrenAndLock()
```

### Reading of the Immediate Hierarchy of a Current Instance

#### Operation

For a node, this action retrieves all or part of the instances of first-level dependent nodes, depending on the instance found in the node's **Instance-presentation** attribute.

This action is valid if:

- The node's Instance-presentation attribute contains an instance.

If this action is valid, its result is the same as that of a read action on an instance and its immediate hierarchy.

This action is always available on root- and dependent-type nodes.

#### Java

- Declaration 

```
public void readFirstChildrenFromCurrentInstance() throws
LocalException, ServerException, CommunicationError,
SystemError
```
- Use name 

```
Read First Children From Detail
```

#### COM

- C++ declaration 

```
public void readFirstChildrenFromDetail()
```
- Internal code 

```
readAllChildrenFromDetail()
```

## Reading of the Complete Hierarchy of a Current Instance

### Operation

This action retrieves all the instances of dependent nodes throughout the Folder, depending on the instance found in the **Instance-presentation** attribute of the root node.

This action is valid if the node's Instance-presentation attribute contains an instance.

If this action is valid, its result is the same as that of a read action on an instance and its complete hierarchy.

This action is always available on all root nodes.

#### Java

- Declaration `public void readAllChildrenFromCurrentInstance() throws LocalException, ServerException, CommunicationError, SystemError`
- Use name `Read All Children From Detail`

#### COM

- C++ declaration `public void readAllChildrenFromDetail()`
- Use name `readAllChildrenFromDetail()`

## Anticipated Reading of an Instance's Immediate Hierarchy

### Operation

This action has the same functionality as a read on an instance and its immediate hierarchy, but allows for an anticipated selection of instances without impacting the GUI.

This action is valid if the instance provided as a parameter does not have an empty value.

If the action is valid, the rules are the same as for the read action on an instance and its immediate hierarchy, except that the Instance-presentation attribute of the affected node may contain an empty value.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

This action is always available on root nodes.

#### Java

- Declaration `public void readFirstChildren({Name of generated class}Data d) throws LocalException, ServerException, CommunicationError, SystemError`
- Use name `Read First Children From {Name of generated class}Data`

#### COM

- C++ declaration `public void readWithFirstChildrenFrom(LPDISPATCH d)`
- Internal code `readWithFirstChildrenFrom({Name of generated class DataDescription} d)`

## Anticipated Reading of an Instance's Complete Hierarchy

### Operation

This action has the same functionality as a read on an instance and its complete hierarchy, but it allows for an anticipated selection of instances without impacting the GUI.

This action is valid if the instance provided as a parameter does not have an empty value.

If the action is valid, the rules are the same as for the read action on an instance and its complete hierarchy, except that the Instance-presentation attribute of the affected node may contain an empty value.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

This action is always available on root- or dependent-type nodes having at least one dependent node.

### Java

- Declaration `public void readAllChildren({Name of generated class DataDescription} d) throws LocalException, ServerException, CommunicationError, SystemError`
- Use name `Read All Children From {Name of generated class}Data`

### COM

- C++ declaration `public void readWithAllChildrenFrom(LPDISPATCH d)`
- Internal code `readWithAllChildrenFrom({Name of generated class DataDescription} d)`

## Management of Paging

### Reading of the Following Page's Instances

#### Operation

This action retrieves the next page in a node's collection. When the selected paging mode is of the **extend** type, retrieved instances are compounded with the instances already present in the **Instance-list-presentation** attribute. Locally-created instances which might conflict with retrieved instances have top priority. When the paging is of the **non-extend** type, instances contained in the **Instance-list-presentation** attribute are overridden with the retrieved instances.

This action is valid if:

- The last page of the collection has not been reached yet. Otherwise, this action does not trigger a server access, and sends a "Collection's last-page retrieval" event.
- On a dependent node, a collection must be already defined, or the **Instance-presentation** attribute of the parent node must contain an instance.

- On a root or reference node, if no collection has been defined, this action operates as an instance-selection action.

If the action is valid:

- The Instance-list-presentation attribute is initialized with the result of the query according to the paging-type and to the collection management mode.
- If the paging applies to a root node and is of the **non-extend** type, with a collection management in automatic mode, the Instance-presentation attribute of the node is set to an empty value.
- If the paging applies to a root or dependent node and is of the **extend** type, or with a collection management in automatic mode, its Instance-presentation attribute as well as the Instance- and Instance-list-presentation attributes of the dependent nodes are not modified.
- The “Total number of local instances” counter is initialized.
- The node-selection return message's label and key are initialized for each dependent node if the collection's last instance has been retrieved.
- A no-error-detection event is sent.
- The event “Presence of updatable local instances” is sent, with a collection management in automatic mode, if updatable instances are still present in the local cache.
- The “Collection's fist-page retrieval” event is sent if the action applies to the root or reference node, if the paging type is **non-extend**, with a collection management in automatic mode, and if the page is the first retrieved page in the collection.
- The “Presence of at least one preceding page” event is sent if the action applies to the root or reference node, if the paging type is **non-extend**, with a collection management in automatic mode, and if the page is not the first retrieved page in the collection.
- The “Presence of at least one following page” event is sent if the last instance of the collection has not been retrieved.
- The “Collection's last-page retrieval” event is sent if the last instance of the collection has been retrieved.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes –if they are present– are set.

This action is always available on all nodes.

## Java

- Declaration `public void readNextPage() throws LocalException, ServerException, CommunicationError, SystemError`
- Use name `Read Next Page`

## COM

- C++ declaration `public void readNextPage()`
- Internal code `readNextPage()`

## Reading of the Preceding Page's Instances

### Operation

This action retrieves the previous page of a node's collection. It is designed exclusively for **non-extend**-type paging with a collection management in automatic mode. Instances found in the Instance-list-presentation attribute are systematically overridden by retrieved instances.

This action is valid if:

- The last page of the collection has not been reached yet. Otherwise, this action does not trigger a server access, and sends the "Collection's first-page retrieval" event.
- If no collection is defined, this action operates like an instance-selection action.

If the action is valid:

- The Instance-list-presentation attribute is initialized.
- The Instance-list-presentation attribute is modified.
- The "Total number of local instances" counter is initialized.
- The selection-return message's label and key are initialized if the collection's last instance has been retrieved.
- The Instance- and Instances-list-presentation attributes of dependent nodes are set to empty values.
- A no-error-detection event is sent.
- The "Presence of updatable local instances" event is sent if updatable instances are still present in the local cache.
- The "Presence of at least one preceding page" event is sent if the page is not the first one retrieved in the collection.
- The "Retrieval of a collection's first page" event is sent if the page is the first one in the collection.
- The "Presence of at least one following page" event is sent if the collection's last instance was not retrieved.
- The "Last page retrieval" event is sent if the collection's last instance was retrieved.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes –if they are present– are set.

The action is available on root- and reference-type nodes.

#### Java

- Declaration `public void readPreviousPage() throws LocalException, ServerException, CommunicationError, SystemError`
- Use name `Read Previous Page`

#### COM

- C++ declaration `public void readPreviousPage()`
- Internal code `readPreviousPage()`

## Sending of Updates

### Sending of Local Updates to the Server

#### Operation

This action sends to the server all the updates performed locally since it was last executed.

Only useful transactions are sent.

This action is valid if at least one local update has been performed.

If the action is valid:

- All updated instances are deleted from the local cache.
- Each modified Logical View instance is updated in the local cache with its last, post-update server image, if the instance-refresh option is set when sending the action.
- Data checking on the server may be activated by setting the appropriate attribute before executing the action.
- A no-error-detection event is sent.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes –if they are present– are set.

This action is available on a root node when at least one of the Elementary Components associated with the Folder's nodes can perform updates.



## Java

- Declaration `public void updateFolder() throws LocalException, ServerException, CommunicationError, SystemError`
- Use name `Update Folder`

## COM

- C++ declaration `public void updateFolder()`
- Internal code `updateFolder()`

## Management of Logical Locks

### Logical Locking of a Current Instance

#### Operation

This action appropriates a Folder instance for exclusive update. It can apply to a local instance identifier representing an instance which does not yet exist in the Database.

This action is valid if:

- The root node's Selection Criteria attribute contains a Logical View instance identifier.
- The instance is not currently locked.

If the action is valid:

- A no-error-detection event is sent.
- The Folder-lock identifier attribute is initialized with the value returned by the server.
- The Folder changes to the “modifiable” status.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.
- The Folder-lock identifier attribute is set to an empty value.
- The Folder changes to the “not-modifiable” status.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes –if they are present– are set.

This action is available on a root node when the logical lock option is set for this Folder in the VisualAge Pacbase Repository.

## Java

- Declaration `public void lock() throws LocalException, ServerException, SystemError, CommunicationError`
- Use name `Lock`

## COM

- C++ declaration `public void lock()`
- Internal code `lock()`

## Logical Unlocking of a Current Instance

### Operation

This action “frees” a Folder instance used for exclusive update when the user chooses not to send locally-updated instances to the server.

This action is valid if:

- The root node's Selection Criteria attribute contains a Logical View instance identifier.
- The instance is locked.

If the action is valid:

- A no-error-detection event is sent.
- The Folder-lock identifier key attribute is set to an empty value.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes – if they are present– are set.

This action is available on a root node when the logical lock option is set for this Folder in the VisualAge Pacbase Repository.

### Java

- Declaration `public void unlock() throws LocalException, ServerException, SystemError, CommunicationError`
- Use name `Unlock`

### COM

- C++ declaration `public void unlock()`
- Use name `unlock()`

## Management of Dependent Instances

### Check on the Presence of Dependent Instances

#### Operation

This action finds out if the Logical View instance contained in the node's Instance-presentation attribute has dependent instances. If this instance was not created locally, and does not contain any dependent instances locally, the system sends this action to the server in order to check for the existence of first-level dependent instances.

This action is valid if the Instance-presentation attribute contains a non-empty value.

If the action is valid:

- A no-error-detection event is sent.
- A "Presence of a dependent instance" or "Absence of dependent instances" event is sent.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

In all circumstances, if the action was passed on to the server:

- Conversation time counters are set.
- Contextual information attributes –if they are present– are set.

This action is available when the node on which it is performed has at least one dependent node.

#### Java

- Declaration `public void checkExistenceOfDependentInstances() throws LocalException, ServerException, CommunicationError, SystemError`
- Use name `Check Existence Of Dependent Instances`

#### COM

- C++ declaration `public void checkExistenceOfDependencies()`
- Internal code `checkExistenceOfDependencies()`

## Management of User Services

### Execution of User Services

#### Operation

This action executes a User Service associated with a node and to its dependent nodes for which a "User Service to be executed" is set.

This action operates if at least one of the relevant nodes contains a non-empty value in the "User Service to be executed" attribute.

If the action is valid:

- A no-error-detection event is sent.
- The presentation attribute of instances returned by a User Service is initialized.
- The "Number of Logical View instances processed by a User Service" attribute is recalculated.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

In all circumstances:

- Conversation time counters are set.
- Contextual information attributes – if they are present – are set.

This action is available if the Elementary Component associated with the node contains at least one User Service.

#### Java

- Declaration `public void executeUserService() throws ServerException, CommunicationError, SystemError, LocalException`
- Use name `Execute User Service`

#### COM

- Declaration `public void executeUserService()`
- Internal code `executeUserService()`

## Management of Asynchronous Conversations

### Deferred Retrieval of a Reply

#### Operation

This action retrieves the reply associated with a query sent with the asynchronous communication type.

This action is valid if:

- The communications protocol used to send the query supports asynchronous conversations.
- The conversation type is asynchronous.
- The identifier of the query specified as parameter is valid and known.

If this action is valid and the query available:

- The rules applied are the same as those governing the action which sent the query –if the query was executed in synchronous mode.
- The “Number of pending replies” attribute is decremented by 1.
- Contextual information attributes –if they are present– are set.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

If the query is not available, the “Unavailable-reply retrieval” event is sent.

In all circumstances, conversation time counters are set.

This action is always available on a root node.

#### Java

- Declaration `public Boolean  
getReply (com.ibm.vap.generic.ServerActionContext  
aContext) throws LocalException, ServerException,  
CommunicationError, SystemError`
- Use name `Get Reply`

#### COM

- C++ declaration `public BOOL getReply(LPDISPATCH i)`
- Internal code `Boolean getReply(<Foldername>ServerAction)`

## Check on a Message Identifier's Validity

### Operation

This action finds out if a query identifier is valid and known.

If the action is valid, the action returns **true** if the key is valid, or **false** otherwise.

If the action is not valid:

- The error is added to the Error Object.
- An error event is sent according to the type of error.

If the query is not available, the "Unavailable-reply retrieval" event is sent.

This action is always available on a root node.

### Java

- Declaration `public Boolean isReplyValid(com.ibm.vap.generic.ServerActionContext aContext)`
- Use name `Checks the validity of the request reply`

### COM

- C++ declaration `public BOOL isReplyValid(LPDISPATCH i)`
- Internal code `Boolean isReplyValid(<Foldername>ServerAction)`

## Sub-Schema Management

### Operation

This action retrieves, by calling the Elementary Component associated with the Logical View, the values of the Data Elements which do not belong to the sub-schema selected via the **subSchema** attribute.

Upon the correct return of this action, the instance is considered to be complete, so its associated implicit sub-schema is reset. Any subsequent modification is then performed with no associated sub-schema.

Before this action is executed, the Data Elements which belong to the sub-schema may have been modified locally.

This action is available if the Elementary Components manage the presence of Data Elements (**VECTPRES=YES** or **CHECKSER=YES**).

### Java

- Declaration `public completeInstance throws LocalException, ServerException, CommunicationError, SystemError`
- Internal code `Complete Instance`

### COM

- C++ declaration `public completeInstance()`
- Use name `completeInstance()`

## Test of Communication with the Server

### Operation

This action enables to perform a test of communication with the server and thus to validate the communication parameters without accessing elementary servers.

#### Java

- Declaration `public void ping() throws CommunicationError`
- Use name `Ping`

#### COM

- C++ declaration `public void ping()`
- Internal code `ping()`

---

## Externalization of the Management of Requests

### Operation

The externalization of the management of requests consists in creating a storage context for actions which are to be executed on the server and thus enables to post different services requests in only one request to the server.

This storage context is defined via a specific object which is an instance of the "MainRequest" class. This object is used to "store" and "post" services requests sent by one Proxy or more before sending them in one same request to the server. Proxies then share the same execution context when updates are requested on several Folders.

## Creation of a Request

### Operation

This action enables to start a storage context for actions which are to be executed on the server by creating an instance of the "MainRequest" class for the Proxy. This action values the "request" attribute for the Proxy. The whole set of actions to be executed are first stored on local in this request object.

#### Java

- Declaration `public void createRequest()`
- Use name `Create Request`

#### COM

- C++ declaration `public void createRequest()`
- Internal code `createRequest()`

## Execution of the Request Actions on the Server

### Operation

This action enables to execute on the server the whole set of actions stored in the request.

This action is available on the MainRequest object.

#### Java

- Declaration `public void sendRequest() throws ServerException, LocalException, CommunicationError, SystemError`
- Use name `Send Request`

#### COM

- C++ declaration `public void sendRequest()`
- Internal code `sendRequest()`

## Cancellation of the Request Actions

### Operation

This action enables to cancel the whole actions which were stored in the request.

This action is available on the MainRequest object.

#### Java

- Declaration `public void cancel()`
- Use name `Cancel`

#### COM

- C++ declaration `public void cancel()`
- Internal code `cancel()`

---

## XML flows handling

The XML flow represents :

- the information hierarchy for a node of the folder (Detail)
- the collection of instances for a node of the folder (Rows)

In both cases, the XML flows correspond to the hierarchical structure of the folder, from the folder root.

For each represented instance, the flow must describe :

- the instance status in the proxy cache,
- the value of the instance fields, (all the fields may not be set as the notion of sub-schema, indicated in the selection operations of proxy instances, is taken into account).



## Status of a Logical View Instance

This notion is similar to the « attribute » notion, defined for a **complexType** (the node) in a XML schema. This attribute can have the values: 'read', 'created', 'modified', 'deleted', or 'unknown' (case of retrieval of invalid/unknown detail instance).

To guarantee the unicity of this attribute name or avoid conflicts with the name of the Logical View fields, this name is on more than 6 characters: **dataStatus**.

## Obtaining of a XML flow

### Operation

This action is used to obtain :

- the instances of a node with hierarchy or not (rows),
- an instance of node with hierarchy or not (detail).

The passed parameters allow:

- to identify the XML flow,
- to identify the node for which you want to retrieve the instance,
- to precise if you want to retrieve rows (**true**),
- to indicate if you want to retrieve the instance of node and its hierarchy (**true**).

This action is not valid if the navigation context of the cache is inconsistent.

If this action is valid :

- If 'hierarchy', backup of the cache navigation context,
- XML flow built by browsing the hierarchy in the cache,
- If 'hierarchy', restoration of the cache navigation context.

If the action is not valid :

- The error is added to the Error Object,
- An error event is sent according to the type of error.

### Java

- Declaration 

```
public void getXML(java.io.OutputStream outputStream, ProxyLv proxy, Boolean rows, Boolean hierarchicalStructure) throws XMLWrapperException
```

### COM

- C++ declaration 

```
public void getXML(utfostream out, ProxyLv proxy, bool rows, bool hierarchicalStructure)
```

## Update from a XML flow

### Operation

In that case, whatever the XML flow, this latter must be valid:

- rows with hierarchy or not,
- detail with hierarchy or not.

The status associated with each node instance in the XML flow guides the cache update. This operation is performed depending on the initial status of the instance in the cache.

Initial status of the instance in the cache	Status of the instance in the XML flow	Local action on the proxy
created	created	modify instance if update
read	created	exception
modified	created	exception
unknown	created	create instance
deleted	created	create instance
created	read	exception
read	read	Nothing
modified	read	exception
unknown	read	Initialize instance
deleted	read	exception
created	modified	Modify instance
read	modified	modify instance
modified	modified	modify instance
unknown	modified	initialize + modify instance
deleted	modified	exception
created	deleted	delete instance
read	deleted	delete instance
modified	deleted	delete instance
unknown	deleted	initialize + delete instance
deleted	deleted	Nothing

**Note:** The lock management is the user's responsibility. It must be performed, if needed, before the cache update from a XML flow.

This action is not valid if the cache navigation context is inconsistent.

If the action is valid:

- SAX-type parsing of the XML flow,
- For each node instance, identification of the associated status: comparison with the status of this instance in the cache before the update.

If this action is not valid:

- The error is added to the Error Object,
- An error event is sent according to the type of error.

#### Java

- Declaration `public void updateFromXML(java.io.InputStream xmlStream) throws XMLWrapperException`

#### COM

- C++ declaration `public void updateFromXML(utfostream xmlStream)`

---

## Management of Proxy Context

### Initialization of the proxy context

#### Operation

This action memorizes the current context and restores an instance of the initialized object. The attributes memorized in this context are: the communication attributes, the extraction and paging attributes, the local updates attributes, the user services attributes, the attributes of the user buffer and of the current instance.

If this action is not valid:

- The error is added to the Error Object,
- A local error event is sent.

This action is available on a root node.

#### Java

- Declaration `public com.ibm.vap.generic.ProxyContext getProxycontext() throws LocalException`
- Use name `getProxyContext`

#### COM

- C++ declaration `public VapTools.VapProxyContext getProxycontext()`
- Internal code `getProxyContext`

## Initialization of the local cache

### Operation

This action resets the attributes in the local cache of the proxy and its dependent nodes from the ProxyContext instance passed as parameter and so the reads and updates can continue.

If the action is not valid:

- The error is added to the Error Object,
- A local error event is sent.

This action is available on a root node.

#### Java

- Declaration 

```
public void  
initFromProxycontext (com.ibm.vap.generic.ProxyContext  
context)  
throws LocalException
```
- Use name 

```
initFromProxycontext
```

#### COM

- C++ declaration 

```
public void initFromProxycontext (VapTools.VapProxyContext  
context)
```
- Internal code 

```
initFromProxycontext
```

## Retrieval of the Proxy

### Operation

This method retrieves the name of the proxy with which the ProxyContext object has been initialized, to re-instantiate it.

#### Java

- Declaration 

```
public String getProxyClassName ()
```
- Use name 

```
getProxyClassName
```

#### COM

- C++ declaration 

```
public BSTR getProxyClassName ()
```
- Internal code 


```
getProxyClassName
```

---

## Chapter 4. Events

---

### Management of Paging

-  For COM, all the events described in the chapter are saved in a stack (consult the paragraph dedicated to the *Management of Events*).

#### Signal of Retrieval of a Collection's Last Page

##### Sending rules

This signal is sent by a node when an instance selection action or a paging action returns a page containing the last instance of the collection. This event is available for root and reference nodes, and for dependent nodes with a maximum cardinal value of n.

##### Java

- Code `noPageAfter`

##### COM

- Code `NO_PAGE_AFTER`

#### Signal of Retrieval of a Collection's First Page

##### Sending rules

This signal is sent by a node when an instance selection action or a paging action returns a page containing the first instance of the collection. This event is available for root and reference nodes, when the paging mode is of the **non-extend** type with a collection management in automatic mode.

##### Java

- Code `noPageBefore`

##### COM

- String code `NO_PAGE_BEFORE`

## Signal of Presence of at Least One Following Page

### Sending rules

This signal is sent by a node when an instance selection action or a paging action returns a page which does not contain the last instance of the collection. This event is available for root and reference nodes, and for dependent nodes with a maximum cardinal value of n.

#### Java

- Code `pageAfter`

#### COM

- Code `PAGE_AFTER`

## Signal of Presence of at Least One Preceding Page

### Sending rules

This signal is sent by a node when an instance selection action or a paging action returns a page which does not contain the first instance of the collection. This event is available for root and reference nodes, when the paging mode is of the `non-extend` type and with a collection management in automatic mode.

#### Java

- Code `pageBefore`

#### COM

- Code `PAGE_BEFORE`

---

## Management of Unit Reads

### Signal of Reading of a Record not Found

#### Sending rules

This signal is sent by a node when an instance-reading action has not returned the required instance.

This action is always available for all nodes.

#### Java

- Code `notFound`

#### COM

- Code `NOT_FOUND`

---

## Management of Simultaneous Selections

### Signal of Non-Participation to a Simultaneous Read

#### Sending rules

This signal is sent by a node following a write/read action the node did not participate to.

This action is always available for all nodes.

#### Java

- Code `notRead`

#### COM

- Code `NOT_READ`

---

## Management of Logical Locks

### Signal of Assigned Logical Lock

#### Sending rules

This signal is sent by a root node following a valid action of logical-lock request.

This action is available for a root node when the logical-lock option is coded for this node in the VisualAge Pacbase Repository.

#### Java

- Code `lockSuccessful`

#### COM

- Code `LOCK_SUCCESSFUL`

### Signal of Unsuccessful Logical Lock

#### Sending rules

This signal is sent by a root node following an aborted action of logical-lock on a instance, this instance being already set to exclusive update mode for another user.

This action is available for a root node when the logical-lock option is coded for this node in the VisualAge Pacbase Repository.

#### Java

- Code `lockFailed`

#### COM

- Code `LOCK_FAILED`

---

## Management of Dependent Instances

### Signal of Presence of at Least One Dependent Instance

#### Sending rules

This signal is sent by a node following a check action on the presence of dependent instances, when the instance concerned has at least one dependent instance. This action is always available for root- and dependent-type nodes.

#### Java

- Code `dependentInstances`

#### COM

- Code `DEPENDENT_INSTANCES`

### Signal of Absence of Dependent Instances

#### Sending rules

This signal is sent by a node following a check action on the presence of dependent instances when the instance concerned has no dependent instances. This action is always available for root- and dependent-type nodes.

#### Java

- Code `noDependentInstances`

#### COM

- String code `NO_DEPENDENT_INSTANCES`



---

## Chapter 5. Public Interface for Data Elements Handling

---

### Management of a Data Element's Contents

#### Description

This attribute displays the Data Element's contents.

This attribute is always available for Data Elements defined in the `DataDescription`, `SelectionCriteria`, and `UserContext` classes.

#### Java

- Type Depends on the type of Data Element (`java.lang.String`, `int`, `long`, `double`, or `java.util.Date`)
- Internal code `<DataElementCode>`
- Use name `<Data Element Clear Name>`
- get/set 

```
public [Type] get<DataElementCode> ()
public void set<DataElementCode> (T-type)
```

#### COM

- Type Depends on the type of Data Element
- Internal code `<DataElementCode>`
- get/set C++ 

```
public [Type] get<DataElementCode> ()
public void set<DataElementCode> (T-type)
```

---

### Management of Authorized-Value Codes

#### Description

This attribute provides the authorized values associated with a Data Element.

This attribute is always available for Data Elements containing authorized values and defined in the `DataDescription` class.

#### Java

- Type `[ Type [ ] ]`
- Internal code `<DataElementCode>Values`
- Use name `<DataElementCode> Values`
- get/set 

```
public [Type[ ] ] get<DataElementCode>Values ( )
set not available
```

#### COM

- Available with a browsing API for collection-type attributes.
- Nb of elements `public long <DataElementCode>ValidValuesCount()`
- Element `public Type <DataElementCode>ValidValuesAt(long i)`

---

## Management of Authorized Values

### Description

This attribute displays the labels of authorized values associated with a Data Element.

This attribute is always available for Data Elements which contain authorized values and which are defined in the `DataDescription` class.

#### Java

- Type `String [ ]`
  - Internal code `<DataElementCode>Labels`
  - Use name `<DataElementCode> Labels`
  - get/set `public String[ ] get<DataElementCode>Labels ( )`  
set not available
- These methods can accept the target local (new parameter which corresponds to the Language for which you want to find the label).

#### COM

- Nb of elements Available with a browsing API for collection-type attributes.  
`public long <DataElementCode>ValidLabelsCount ( )`
- Element `public Type <DataElementCode>ValidLabelsAt (long i)`

---

## Management of the Validity of a Data Element's Contents

### Description

This action specifies whether the contents of a Data Element are valid or not.

This action is always available for Data Elements which contain authorized values and which are defined in the `DataDescription` class.

#### Java

- Declaration `public DataFieldError get<DataElementCode>Error ( )`
  - Use name `<DataElementCode> Error`
- This method returns a `DataFieldError` instance which indicates the nature of the error detected on the field or the `null` value if the Data Element contents are empty.

#### COM

Not available.

---

## Access to the Characteristics of a Data Element

### Operation

The `DataGroup` class offer methods which enable to retrieve the input characteristics of Data Elements fields composing a Logical View.

- `findDataFieldFormat`: sent back the Data Element VA Pac format. (Sends nothing back for alphanumeric Data Element),
- `findDataFieldMaxLength`: sent back the maximum length authorized for the value of a Data Element.

These actions are available for all the Data Elements which are defined in the `DataGroup` class.

#### Java

- Declaration `public java.text.Format findDataFieldFormat(String code)`
- Use name `findDataFieldFormat`

#### COM

Not available.

#### Java

- Declaration `public int findDataFieldMaxLength (String code)`
- Use name `findDataFieldMaxLength`

#### COM

Not available.

---

## Initialization of the Data Elements Values

### Operation

This method enables to initialize the values of a `DataGroup` instance from the values of an other `DataGroup` instance.

#### Java

- Declaration `public void initializeFrom(DataGroup)`
- Use name `Initialize From`

#### COM

Not available.

---

## Management of a Data Element's Presence

### Description

These actions specify whether the Data Element is absent (empty contents) or present (contents not empty).

These actions are always available for Data Elements defined in the `DataDescription` and `UserDataDescription` classes

Before this action is executed, all Data Elements are considered to be absent, except if a default value has been specified in VisualAge Pacbase.

#### Java

- Declaration `public boolean is<DataElementCode>Present ()`  
`public void set<DataElementCode>Present (boolean b)`
- Use name `<DataElementCode> Present`

#### COM

- C++ declaration `public bool is<DataElementCode>Present ()`  
`public void set<DataElementCode>Present (bool b)`
- Internal code `Boolean is<DataElementCode>Present/`  
`setDataElementCodePresent (Boolean value)`

---

## Management of a Data Element's Check

### Operation

These actions specify whether the Data Element is to be checked or not.

These actions are always available for Data Elements defined in the `DataDescription` and `UserDataDescription` classes of the root or dependent nodes whose Elementary Component includes the `NULLMNGT=YES` and `CHECKSER=YES` options and an update service.

Before this action is executed, all Data Elements are to be checked (if the `serverCheckOption` attribute is set to true).

#### Java

- Declaration `public setCheck(int index, boolean aBoolean)`
- Use name `Check Flag for the Index's field`

The index of the Data Element to be checked is found via the following method:

- Declaration `public int get<DataElementCode>Index()`
- Use name `<DataElementCode> Index`

#### COM

- C++ declaration `public void setCheck(long fieldIndex, BOOL b)`
- Internal code `setCheck (Long, Boolean)`

---

## Management of membership to a Sub-schema

### Data Element Belonging to the Sub-Schema

#### Operation

This action enables you to know whether the Data Element whose index passed as a parameter belongs to the sub-schema associated with the instance contained in the **detail** attribute.

This action is available if the Elementary Components manage the presence of Data Elements (**VECTPRES=YES** or **CHECKSER=YES**) and if the node includes at least one sub-schema.

#### Java

- Declaration `public boolean belongsToSubSchema(int indexDataElement)`
- User name `Belongs to current subschema`

#### COM

- C++ declaration `public BOOL belongsToSubSchema(short index)`
- Internal code `Boolean belongsToSubSchema(Integer)`

---

## Management of membership to an extraction method

This action enables you to know whether the Data Element passed as a parameter belongs to the extraction method, which also passed as a parameter.

This action is available in the **SelectionCriteria** classes whose node includes at least one extraction method.

#### Java

- Declaration `public boolean dataFieldBelongsToExtractMethod (String FieldCode, String extractMethodCode)`
- Internal code `DataFieldsBelongsToExtractMethod`

#### COM

- C++ declaration `public BOOL dataFieldBelongsToExtractMethod (LPCTSTR FieldCode, LPCTSTR extractMethodCode)`
- Internal code `Boolean dataFieldBelongsToExtractMethod(String, String)`



---

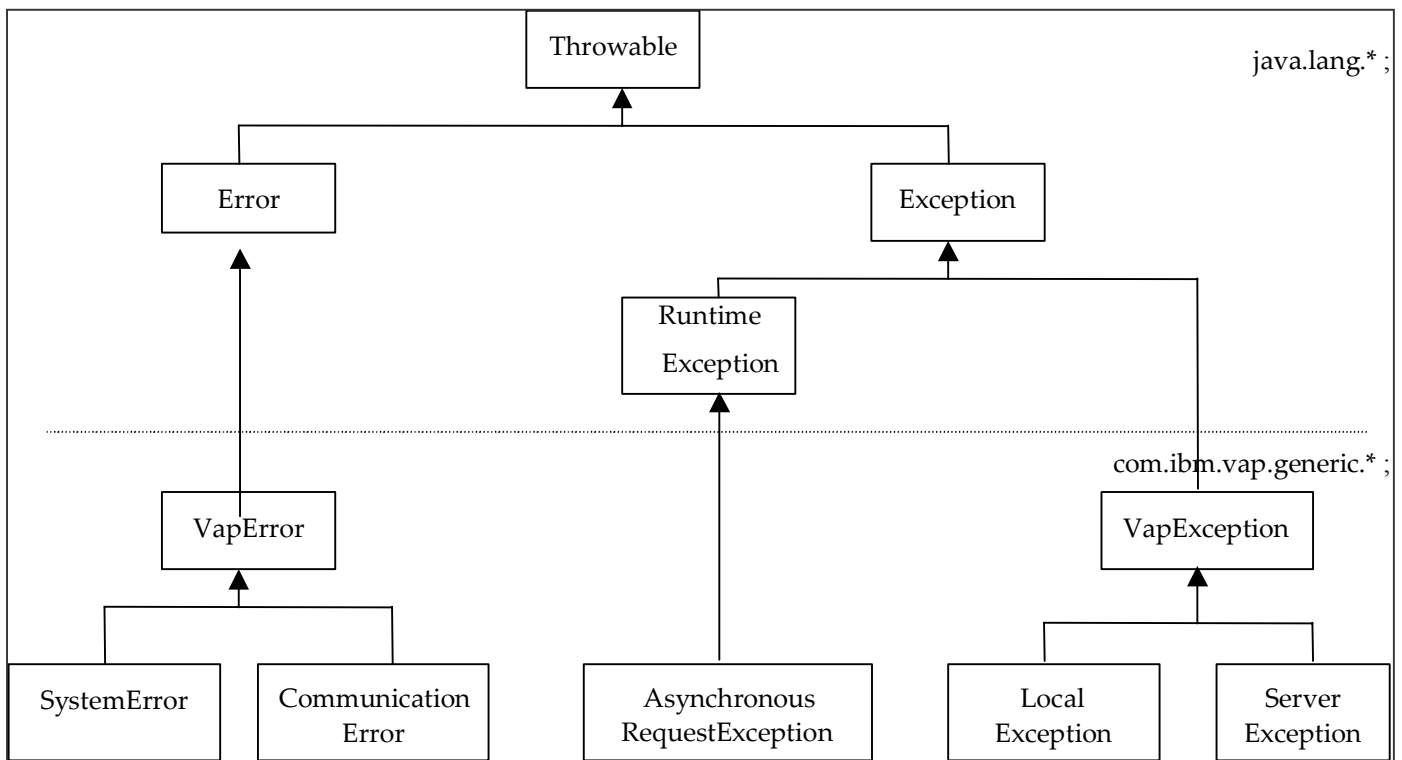
## Chapter 6. Management of Errors

---

### Management of Errors for Java Target

The management of the errors associated with the handling of Java Proxies is based on the raise of exceptions mechanism.

Four classes enable to carry errors or exceptions issued from a VisualAge Pacbase Proxy. The four classes are available to the developer and they inherit from `java.lang.Throwable` in the following way:



The `com.ibm.vap.generic.AsynchronousRequestException` exception is raised at a server access whenever the Proxy is in asynchronous mode.

The execution of some methods of the Proxy require the control of any exception inheriting from `com.ibm.vap.generic.VapException` (refer to the declarations documented in the manual to know the different types of exceptions that can be sent by a method).

It is also highly recommended to control errors inheriting from `com.ibm.vap.generic.VapError` although the control is optional for this type of class in Java.

**Note** An excessive use of some Proxy methods may also raise the `java.lang.IllegalStateException` exception, like in the case of the use of a method which does not match with the VisualAge Pacbase Proxy Definition (example: using the `createUserInstance` method although no user service was defined in the VisualAge Pacbase Server Description).

## Classes Related to the Management of Errors

### Communication Errors

#### Name of the exception

`com.ibm.vap.generic.CommunicationError`

**Note** The exception is raised if an error occurs in the communication string with the server.

The message carried by each instance of the class gives information concerning the detected error (the `getMessage()` method of the class).

### System Errors

#### Name of the exception

`com.ibm.vap.generic.SystemError`

**Note** The exception is raised by system errors.

This type of error represents an internal and irretrievable error. It can be detected either by the client or by the server.

If the server detects system errors, messages associated with these errors are represented by instances of `com.ibm.vap.generic.ServerMessage`. They are available through the `java.util.Enumeration serverMessages()` method from the `com.ibm.vap.generic.SystemError` class.

☞ Refer to the *eBusiness & Pacbench C/S Applications - Graphic Presentation* guide for the list of the system errors.

### Local Errors

#### Name of the exception

`com.ibm.vap.generic.LocalException`

**Note** The exception is raised by local errors. These errors are detected by the client.

The exception holds a property of the `int` type (`int getLocalExceptionKey()`) that enables to identify the type of error leading to the raise of the exception (mistaken creation, invalid instance, ...).



Errors responsible for the exception are described in the *eBusiness & Pacbench C/S Applications - Graphic Presentation* guide and also in the HTML documentation associated with the generic classes: Package `com.ibm.vap.generic`.

## Server Errors

### Name of the exception

`com.ibm.vap.generic.ServerException`

**Note** The exception is raised by server errors. It is raised upon the reception of logical error message(s) detected by the server. These logical error messages are represented by instances of `com.ibm.vap.generic.ServerMessage` class. The list of error messages received from the server is available via the `java.util.Enumeration serverMessages()` method from the `com.ibm.vap.generic.ServerException` class.

In the case of errors detected at the update service, it is possible to restore the context of the Proxy related to the update request (restoration of the selection tree and of the detail) for each error.

The `com.ibm.vap.generic.ServerException` class indicates if there are restorable errors in the list of errors detected by the method (`boolean isContextRestorable()`).

### Error Messages Received from the Server

Error messages received from the server are represented by objects with the `com.ibm.vap.generic.ServerMessage` type.

This interface offers methods enabling also to fetch the error key (`String key()`), the error message label (`String label()`) and the error message label formatted by the Client (`String localLabel()`).

For the description of the local formatting principle for error message labels, refer to section *Customizing Error Messages*, on page 102.

### Error Messages Received from the Server on the Update

This interface inherits from `com.ibm.vap.generic.ServerMessage` interface.

Objects of this type represent error messages received from the server; these errors have been detected at the execution of update services.

This interface offers methods allowing to know and to restore the context of the Proxy related to the update request:

- `boolean isContextRestorable()`: indicates if the error context is « restorable »,
- `void restoreContext() throws LocalException`: triggers the restoration of the context related to the error update,
- `DataDescription erroneousData()`: returns the `DataDescription` class for which the update failed,

- `HierarchicalProxyLv erroneousProxy()`: returns the `HierarchicalProxy` class that handles the update request which failed.

## Customizing Error Messages

Customizing message labels for errors associated with the handling of Proxies is possible with Java Proxies. The customization is based on the use of internationalization and dynamic formatting techniques of labels provided by Java language: the different labels are stored in a resource file (`vaperror.properties`) that is loaded according to the geographical context (the context being provided by the default `Local`). The file is structured on a key-value relation mode where the value corresponds to a label.

Also, each label must respect the formality of the « *patterns* » used by Java when handling labels with variable parts (`java.text.MessageFormat`).

The error labels stored in the file are:

- the message associated with an instance of `com.ibm.vap.generic.LocalException`, `SystemError` and `ServerException` (available through method `String getMessage()`),
- the label of objects with `com.ibm.vap.generic.ServerMessage` type locally formatted (`String localLabel()`).

### Local Error Messages


Keys that enable to find local error messages correspond to the name of the constants defined on the `com.ibm.vap.generic.LocalException` class. The keys represent the different types of local errors and are prefixed with `LOCAL_`.

*Example* : the key which enables to find the label associated with a local exception of the `INVALID_INSTANCE` type will be `LOCAL_INVALID_INSTANCE`.

### Local Error Messages Received from the Server Component

Error messages received from the server are set up with two information: a key and a label.

The key of the error message is read in order to identify the key of storage of the local label associated with the error.

 The structure of the error message key is described in the *eBusiness & Pacbench C/S Applications - Graphic Presentation* guide.

In the case of a system error, the access key to the local label corresponds to the `SYSTEM_` prefix followed by characters comprised between columns 14 and 19 if they have a significant value (not blank) or followed by characters comprised between columns 10 and 13, if the value is not significant.

In the case of error servers (user errors, for instance), the access key to the local label corresponds to the characters comprised between columns 14 and 19 if they have a significant value and if the character in column 20 is blank. If the character of column 20 is 2 or 5, the error key is respectively `REQUIRED` and `VALUE`. If the characters comprised between columns 14 and 19 have no significant value, the access key to the local label corresponds to the character comprised between columns 22 and 25.

## Server and System Error Messages

The label of the `com.ibm.vap.generic.ServerException` -type exceptions and of the `com.ibm.vap.generic.SystemError` -type errors are respectively available with the `VAP_SERVER_EXCEPTION` and `VAP_SYSTEM_ERROR` keys.

### Example of Error Messages File

```
# This file defines the default error labels in VisualAge Pacbase for Java.
# Labels stored in this file are defined after the java.text.MessageFormat
pattern.
# The possible arguments in the labels are :
# {0} = library
# {1} = server name
# {2} = view code
# {3} = data id
# {4} = attribute name
# {5} = attribute value
# {6} = technical label (technical message from the local cache or message
label from the server)
# Note: Those arguments are filled within the error context. They may not have
a value if the argument is not meaningful in the error context.

# Local Exception Error Messages
LOCAL_PARENT_INSTANCE_MISSING = Parent instance missing (data id: {3})
LOCAL_CURRENT_INSTANCE_MISSING = Current instance missing
LOCAL_SERVER_UPDATE_REQUIRED = Server update required (data id: {3})
LOCAL_UNKNOWN_INSTANCE = Unknown instance (data id: {3})
LOCAL_INVALID_INSTANCE = Invalid instance
LOCAL_INSTANCE_NOT_LOCKED = Instance not locked (data id: {3})
LOCAL_INVALID_CREATION = Invalid creation (data id: {3})
LOCAL_INVALID_CHANGE = Invalid change (data id: {3})
LOCAL_INVALID_DELETION = Invalid delete (data id: {3})
LOCAL_INVALID_INITIALIZATION = Invalid initialization (data id: {3})
LOCAL_CARDINALITY_VIOLATION = Cardinality violation {6} (data id: {3})
LOCAL_INSTANCE_ALREADY_LOCKED = Already locked instance (data id: {3})
LOCAL_CURRENT_USER_INSTANCE_MISSING = Current user instance missing
LOCAL_REFERRING_INSTANCE_MISSING = Referring instance missing
LOCAL_ASYNCHRONOUS_VIOLATION = Asynchronous violation ({6})
LOCAL_UNKNOWN_CONTEXT = Unknown context
LOCAL_VALUE_REQUIRED = Required item: {4} (data id: {3})
LOCAL_VALUE_ERROR = Value error: {4} (value: {5}, data id: {3})
LOCAL_LENGTH_ERROR = Length error on instance field: {4} (value: {5}, data id:
{3})
LOCAL_SUBSCHEMA_ERROR = Field {4} is out of sub-schema (value: {5}, data id:
{3})
LOCAL_FOLDER_USER_CONTEXT_LENGTH_ERROR = Length error on instance field in
Folder user context: {4} (value: {5})
LOCAL_REFERENCE_USER_CONTEXT_LENGTH_ERROR = Length error on instance field in
a reference user context: {4} (value: {5})

# Server Service Error Messages
REQUIRED = Required value: {4} (library: {0}, server: {1}, view: {2})
VALUE = Value error: {4} (library: {0}, server: {1}, view: {2})
DUPL = Invalid creation (library: {0}, server: {1}, view: {2})
NFND = Invalid delete or modification (library: {0}, server: {1}, view: {2})
LOCKED = Already locked instance (library: {0}, server: {1})
NTLOCK = Instance not locked (library: {0}, server: {1})

# System Error Messages
SYSTEM_STRU = Structure error onto Logical View (library: {0}, server: {1})
SYSTEM_VERS = Version error (library: {0}, server: {1})
SYSTEM_VIEW = Unknown view (library: {0}, server: {1})
SYSTEM_SERV = Unknown service (library: {0}, server: {1})
SYSTEM_LTH = Length view error (library: {0}, server: {1})
SYSTEM_LSRV = Length received message error (library: {0}, server: {1})
SYSTEM_NUVE = Version error in Elementary Component (library: {0}, server:
{1})
SYSTEM_PCVLTH = Message length error (library: {0}, server: {1})
SYSTEM_MISPCV = Components out of phase
```

```

SYSTEM_ACCESS = Data access error {6} (library: {0}, server: {1}, view: {2})
SYSTEM_LKABSC = Invalid absence of lock processing (library: {0}, server: {1})
SYSTEM_WF00 = Temporary file access error or Database connect error (error :
{6})
SYSTEM_TAND = Pathsend error {6} (library: {0}, server: {1})
SYSTEM_PILO = Pilot Record not found during user buffer processing (library:
{0}, server: {1})
SYSTEM_EXT1 = Extract method : PCV syntax error or size error (library: {0},
server: {1})
SYSTEM_EXT2 = Unknown extract method (library: {0}, server: {1})
SYSTEM_USR1 = User service not found (library: {0}, server: {1})
SYSTEM_USR2 = Size error for user service (library: {0}, server: {1})
SYSTEM_USR3 = Unknown user service (library: {0}, server: {1})

# Internal Exception Labels
VAP_SERVER_EXCEPTION = A Server Exception occurred.
VAP_SYSTEM_ERROR = A System Error occurred

```

---

## Management of Errors for COM Target

In the COM environment, there is a **VAPERROR** interface which contains the attributes, actions and events enabling the consultation of all types of errors (local, communication and server). This interface is available in the VapTools Library delivered with the generator. This library must be saved on the workstation (regsvr32 VapTools.dll) and referenced in the client language before being used.

### Access Method to Errors

#### Operation

This method enables to retrieve an instance of the VapError object sent back by local or server actions.

- C++ declaration `public VapError getErrorsElementAt(int i)`
- Internal code `getErrorsElementAt(long i)`

#### Operation

This method contains the number of errors sent back by a local or server action.

It is available on each node of the Proxy.

- C++ declaration `public Int getErrorsCount()`
- Internal code `getErrorsCount`

## VapError Attributes

### Management of the Error Type

#### Operation

This action enables to retrieve the "LOCAL", "SERVER" or "COMMUNICATION" type of error.

- C++ declaration `public BSTR getType()`
- Internal code `String getType()`

### Management of the Action Which Triggers the Error


#### Operation

This action enables to retrieve the action which triggered the error.

- C++ declaration `public BSTR getAction()`
- Internal code `String getAction()`

### Management of the Error Key


#### Operation

 This action enables to retrieve the error key (refer to the *eBusiness & Pacbench C/S Applications – Graphic Presentation* guide for the list of errors).

- C++ declaration `public BSTR getType()`
- Internal code `String getKey`

### Management of the Error Label

#### Operation

 This action enables to retrieve the default or customized error label (cf. section *Customizing Error Messages* for information on the customization procedure).

- C++ declaration `public BSTR getLabel()`
- Internal code `String getLabel()`

### Management of the Error Gravity

#### Operation

This action enables to retrieve the error gravity, "EXCEPTION" or "ERROR"

- C++ declaration `public BSTR getGravity()`
- Internal code `String getGravity`

## Events Linked to Errors

For the COM target, all the events described in this chapter are to be retrieved in the whole events associated with each Proxy using the `public String popServerEvent()` method as long as `public Int getServerEventsCount()` does not send back zero. The String which are retrieved correspond to the codes of the events described.

### Signal for No Error Detection

#### Sending rules

This signal is sent by the Error Manager when no local, server or system error is detected.

This event is systematically available.

- Code `NO_ERROR`

### Signal for Local Error Retrieval

#### Sending rules

This signal is sent by the Error Manager when a local action detects an error.

This event is systematically available.

- Code `LOCAL_ERROR`

### Signal for Server Error Retrieval

#### Sending rules

This signal is sent by the Error Manager when a server detects an access logical error, a user error or an error on the Logical View data check.

This event is systematically available.

- Code `SERVER_ERROR`

### Signal for System Error Retrieval

#### Sending rules

This signal is sent by the Error Manager when a server detects a severe error such as a discrepancy of the versions between the Client Component and the associated Elementary Component.

This event is systematically available.

- Code `SYSTEM_ERROR`

## Signal for Communication Error Retrieval

### Sending rules

This signal is sent by the Error Manager when a communication error is detected during an exchange between a Client Component and a Server Component.

This event is systematically available.

- Code `FATAL_ERROR`

## Customizing Error Messages

Customizing the messages linked to errors associated with the handling of Proxies is possible with COM Proxies. The various messages are stored in a text file (VapProxyMsg.txt by default) located in a directory that can be accessed by the Path environment variable.

### Naming Rules for Error Messages Files

The following rules apply to the storage of error messages:

- As much VapErrorMsg text files as targeted languages. The name of files is formatted as follows : VapProxyMsg\_ language\_ country.txt.
- You must try first to load the file which corresponds to the geographic context (this context is given by the default **Local**). If this file is not found, try to load the file with the language extension (VapProxyMsg\_ language.txt). If this file is not found, try then to load the default VapProxyMsg.txt file.
- This file is structured following the key-value relation mode where the value corresponds to a label. The error messages stored in this file are messages associated with local, system or server errors.

### Syntax of Error Messages Files

To modify or create a file of error messages, a number of syntax rules must be followed Any syntax error triggers the complete invalidity of the externalization of error messages.

- Use quotes to delimit a key-value relation. This relation must be found on one same line.
- Any line which does not begin with a quote is not read.
- The key and the message must be “framed” separately with quotes.
- To specify a variable attribute in a message, use the { and } curly brackets to enclose the attribute number.

### Error Messages for Local Exceptions

Keys that enable to find local error messages correspond to the names of constants defined for the local exceptions representing the various types of local errors, prefixed by **LOCAL\_**

**Example** : the key which enables to find the error message associated with a local exception whose type is **INVALID\_INSTANCE** is **LOCAL\_INVALID\_INSTANCE**.

## Local Error Messages Received from the Server Component

Error messages received from the Server are set up with two information: a key and a label.

The key of the error message is read in order to identify the key of storage of the local label associated with the error.

☞ The structure of the error message key is described in the *eBusiness & Pacbench C/S Applications - Graphic Presentation* guide.

In the case of system errors, the access key to the local label corresponds to the **SYSTEM\_** prefix followed by characters comprised between columns 14 and 19 if they have a significant value (not blank) or followed by characters comprised between columns 10 and 13, if the value is not significant.

In the case of server errors (user errors, for instance), the access key to the local label corresponds to the characters comprised between columns 14 and 19 if they have a significant value and if the character in column 20 is blank.

If the character of column 20 is 2 or 5, the error key is respectively **REQUIRED** and **VALUE**.

If the characters comprised between columns 14 and 19 have no significant value, the access key to the local label corresponds to the character comprised between columns 22 and 25.

## Server and System Error Messages

The label for Server and System exceptions are respectively available with the **VAP\_SERVER\_EXCEPTION** and **VAP\_SYSTEM\_ERROR** keys.

## Example of Error Message File

```
# This file defines the default error labels in VisualAgePacbase for Java.
# The labels are stored in the bundle are potential java.text.MessageFormat
pattern.
# The possible arguments in the label are :
# {0} = library
# {1} = server name
# {2} = view code
# {3} = data id
# {4} = attribute name
# {5} = attribute value
# {6} = technical label (technical message from the local cache or message
label from the server)
# Note : Those arguments are filled within the error context. They may not
have value if the argument is
#      not meaningful in the error context.

# Local exception error
LOCAL_PARENT_INSTANCE_MISSING = Parent instance missing (data id: {3})
LOCAL_CURRENT_INSTANCE_MISSING = Current instance missing
LOCAL_SERVER_UPDATE_REQUIRED = Server update required (data id: {3})
LOCAL_UNKNOWN_INSTANCE = Unknown instance (data id: {3})
LOCAL_INVALID_INSTANCE = Invalid instance
LOCAL_INSTANCE_NOT_LOCKED = Instance not locked (data id: {3})
LOCAL_INVALID_CREATION = Invalid creation (data id: {3})
LOCAL_INVALID_CHANGE = Invalid change (data id: {3})
LOCAL_INVALID_DELETION = Invalid delete (data id: {3})
LOCAL_INVALID_INITIALIZATION = Invalid initialization (data id: {3})
LOCAL_CARDINALITY_VIOLATION = Cardinality violation {6} (data id: {3})
LOCAL_INSTANCE_ALREADY_LOCKED = Already locked instance (data id: {3})
LOCAL_CURRENT_USER_INSTANCE_MISSING = Current user instance missing
LOCAL_REFERRING_INSTANCE_MISSING = Referring instance missing
LOCAL_ASYNCHRONOUS_VIOLATION = Asynchronous violation ({6})
```



LOCAL\_UNKNOWN\_CONTEXT = Unknown context  
LOCAL\_VALUE\_REQUIRED = Required item: {4} (data id: {3})  
LOCAL\_VALUE\_ERROR = Value error: {4} (value: {5}, data id: {3})  
LOCAL\_LENGTH\_ERROR = Length error on instance field: {4} (value: {5}, data id: {3})  
LOCAL\_SUBSCHEMA\_ERROR = Field {4} is out of subschema (value: {5}, data id: {3})  
LOCAL\_FOLDER\_USER\_CONTEXT\_LENGTH\_ERROR = Length error in instance field of folder user context: {4} (value: {5})  
LOCAL\_REFERENCE\_USER\_CONTEXT\_LENGTH\_ERROR = Length error in instance field of reference user context: {4} (value: {5})  
LOCAL\_REQUEST\_ALREADY\_EXISTS = External request already exists  
LOCAL\_REQUEST\_BAD\_USERBUFFER = Incorrect folder user context for linking the external request ({6})  
LOCAL\_REQUEST\_NOT\_ACTIVE = External request state unallowed for create or link  
LOCAL\_UPDATE\_CURRENTLY\_POSTED = Instance update already posted (data id: {3})  
LOCAL\_NO\_SERVER\_RESPONSE\_EXPECTED = No server response expected  
LOCAL\_LOCK\_SERVICE\_ALREADY\_REQUESTED = Lock service already present in the request for this instance (data id: {3})  
LOCAL\_UNLOCK\_SERVICE\_ALREADY\_REQUESTED = Unlock service already present in the request for this instance (data id: {3})  
LOCAL\_READ\_SERVICE\_ALREADY\_REQUESTED = Same read service already present in the request  
LOCAL\_REQUEST\_BAD\_APPLICATION = Incorrect eBusiness application for linking the external request  
LOCAL\_REQUEST\_TOO\_LARGE = The request has reached its maximum size, can't create more services in it

# Server Service Error Messages  
REQUIRED = Required value: {4} (library: {0}, server: {1}, view: {2})  
VALUE = Value error: {4} (library: {0}, server: {1}, view: {2})  
DUPL = Invalid creation (library: {0}, server: {1}, view: {2})  
NFND = Invalid delete or modification (library: {0}, server: {1}, view: {2})  
LOCKED = Already locked instance (library: {0}, server: {1})  
NTLOCK = Instance not locked (library: {0}, server: {1})

# System Error Messages  
SYSTEM\_STRU = Structure error onto logical view (library: {0}, server: {1})  
SYSTEM\_VERS = Version error (library: {0}, server: {1})  
SYSTEM\_VIEW = Unknown view (library: {0}, server: {1})  
SYSTEM\_SERV = Unknown service (library: {0}, server: {1})  
SYSTEM\_LTH = Length view error (library: {0}, server: {1})  
SYSTEM\_LSRV = Length received message error (library: {0}, server: {1})  
SYSTEM\_NUVE = Version error in business component (library: {0}, server: {1})  
SYSTEM\_PCVLTH = Message length error (library: {0}, server: {1})  
SYSTEM\_MISPCV = Components out of phase  
SYSTEM\_ACCESS = Data access error {6} (library: {0}, server: {1}, view: {2})  
SYSTEM\_LKABSC = Invalid absence of lock processing (library: {0}, server: {1})  
SYSTEM\_WF00 = Temporary file access error or Database connect error (error : {6})  
SYSTEM\_TAND = Path send error {6} (library: {0}, server: {1})  
SYSTEM\_PILO = Pilot Record not found during user buffer processing (library: {0}, server: {1})  
SYSTEM\_EXT1 = Extract method : PCV syntax error or size error (library: {0}, server: {1})  
SYSTEM\_EXT2 = Unknown extract method (library: {0}, server: {1})  
SYSTEM\_USR1 = User service not found (library: {0}, server: {1})  
SYSTEM\_USR2 = Size error for user service (library: {0}, server: {1})  
SYSTEM\_USR3 = Unknown user service (library: {0}, server: {1})

# Internal Exception Labels  
VAP\_SERVER\_EXCEPTION = A Server Exception occurred.  
VAP\_SYSTEM\_ERROR = An System Error occurred.



# Index

## Classes

Data Class Inheritance diagrams .....	3
DataDescription .....	3
DataDescriptionUpdate Class .....	4
Inheritance Diagrams of the ProxyLv class Class .....	5
SelectionCriteria Class .....	4
UserContext Class .....	4

## COM actions

<DataElementCode>ValidLabelsCount() .....	94
<DataElementCode>ValidValuesCount() .....	93
checkExistenceOfDependencies() .....	79
completeInstance .....	82
createInstance .....	48
createUserInstance() .....	54
deleteInstance .....	49
deleteUserInstance() .....	56
executeUserService() .....	80
getDetailFromDataDescription({}LPDISPATCH d) .....	57
getDetailFromRowIndex .....	58
getProxyClassName .....	88
getProxyContext .....	87
getReply(<Foldername>ServerAction) .....	81
getReply(ServerActionContext s) .....	81
getUserDetailFromDataDescription({}LPDISPATCH d) .....	58
initFromProxyContext .....	88
isReplyValid(<Foldername>ServerAction) .....	82
isReplyValid(ServerActionContext s) .....	82
lock() .....	77
MainRequest .....	44, 83, 84
modifyInstance .....	49
modifyUserInstance() .....	55
ping .....	83
readAllChildrenFromDetail() .....	72
readFirstChildrenFromDetail() .....	71
readInstance() .....	68
readInstanceAndLock() .....	68
readInstances() .....	68
readNextPage .....	75
readPreviousPage .....	76
readWithAllChildren() .....	71
readWithAllChildrenAndLock() .....	71
readWithAllChildrenFrom({}LPDISPATCH d) .....	73
readWithFirstChildren() .....	70
readWithFirstChildrenAndLock() .....	70
readWithFirstChildrenFrom({}LPDISPATCH d) .....	72
resetAllRefreshOption() .....	61
resetCollection() .....	59
resetExtractMethodCodes() .....	60
resetSelectionCriteria(s) .....	62
ResetSubSchema .....	65
resetUserRows() .....	61
resetUserServiceCodes() .....	60
selectInstances .....	67
setCheck(Long, Boolean) .....	96
transferReferenceFromSelectedRow({}LPDISPATCH d) .....	63
undoAllLocalFolderUpdates() .....	51
UndoAllLocalUpdate() .....	53
undoLocalUpdate .....	52
undoLocalUpdate(DataUpdate d) .....	50
unlock() .....	78
updateFolder() .....	77

## COM attributes

<DataElementCode> .....	93
accessInfoKey .....	26
accessInfoLabel .....	26
asynchronous .....	40
communicationResponseTime .....	42
detail .....	18

extractMethodCode .....	9
FolderUserContext .....	28
GetErrorCount() .....	105
getGravity .....	105
getKey .....	105
getLabel .....	105
getType .....	105
globalSelection .....	14
host .....	33
is<DataElementCode>Present .....	96
lastReplyContext .....	40
localSort .....	16
location .....	32
LocationsFile .....	34
LockTimestamp .....	25
manualCollectionReset .....	10
maximumReplyCount .....	41
maxNumberOfRequestedInstances .....	13
password .....	33
pendingReplyCount .....	41
port .....	34
referenceUserContext .....	28
refreshOption .....	12
serverCheckOption .....	11
serverResponseTime .....	42
set<DataElementCode>Present(BOOL Boolean Value) .....	96
setProperty .....	36
UserDetail .....	23
userId .....	32
userServiceCode .....	24
VapError getErrorElementAt(Int i) .....	104

## COM events

DEPENDENT_INSTANCES .....	92
FatalError .....	107
LocalError .....	106
LOCK_FAILED .....	91
LOCK_SUCCESSFUL .....	91
NO_DEPENDENT_INSTANCES .....	92
NO_PAGE_AFTER .....	89
NO_PAGE_BEFORE .....	89
NoError .....	106
NOT_FOUND .....	90
NOT_READ .....	91
PAGE_AFTER .....	90
PAGE_BEFORE .....	90
ServerError .....	106
SystemError .....	106

## Java events

dependentInstances .....	92
lockFailed .....	91
lockSuccessful .....	91
noDependentInstances .....	92
noPageAfter .....	89
noPageBefore .....	89
notFound .....	90
notRead .....	91
pageAfter .....	90
pageBefore .....	90

## Java methods

<DataElementCode>Error .....	94
<DataElementCode>Index .....	96
belongsToSubschema .....	97
checkExistenceOfDependentInstances() .....	79
completeInstance .....	82
createInstance() .....	48
createUserInstance() .....	54
deleteInstance() .....	49
deleteUserInstance() .....	56

executeUserService()	80
findDataFieldFormat	95
findDataFieldMaxLength	95
getDetailFromDataDescription({} d)	57
getDetailFromRowIndex	58
getFolderConstants()	64
getNodeConstants()	65
getProxyclassName	88
getProxyContext	87
getReply(com.ibm.vap.generic.ServerActionContext aContext)	81
getUserDetailFromDataDescription({} d)	58
getXML	85
initFromProxyContext	88
initializeFrom	95
initializeInstance	62
isReplyValid(com.ibm.vap.generic.ServerActionContext aContext)	82
lock()	77
MainRequest	44, 84
MainRequest	83
modifyInstance()	49
modifyUserInstance()	55
ping	83
readAllChildren({} d)	73
readAllChildrenFromCurrentInstance()	72
readFirstChildren({}Data data)	72
readFirstChildrenFromCurrentInstance()	71
readInstance()	68
readInstanceAndLock()	68
readInstances()	68
readInstanceWithAllChildren()	71
readInstanceWithAllChildrenAndLock()	71
readInstanceWithFirstChildren()	70
readInstanceWithFirstChildrenAndLock()	70
readNextPage()	75
readPreviousPage()	76
resetAllRefreshOption()	61
resetCollection	59
resetExtractMethodCodes()	60
resetSelectionCriteria()	62
resetSubSchema	65
resetUserRows()	61
resetUserServiceCodes()	60
restoreSelection ({}Data d)	59
selectInstances()	67
setCheck(int index, boolean a Boolean)	96
transferReferenceFromSelectedRow({} d)	63
undoAllLocalFolderUpdates()	51
undoAllLocalUpdate	53
undoLocalFolderUpdates(ClientDataUpdate d)	50
undoLocalUpdate	52
unlock()	78
updateFolder()	77
updateFromXML	87
<b>Java properties</b>	
<DataElementCode>	93
<DataElementCode>Labels	94
<DataElementCode>Values	93
accessInfoKey	26
accessInfoLabel	26
asynchronous	40
communicationResponseTime	42
dataComparator	17
detail	18
extractMethodCode	9
extractMethodCodes	8
FolderInstancesCount	29
FolderUpdatedInstancesCount	29
FolderUserContext	28
globalSelection	14
host	33
iRows	15
is<DataElementCode>Present	96
iUpdatedFolders	19
iUpdatedInstances	20
iUserInputRows	21
iUserOutputRows	22
lastReplyContext	40
localSort	16
location	32
locations	31
lockTimestamp	25
manualCollectionReset	10
maximumNumberOfRequestedInstances	13
maximumReplyCount	41
nodeUpdatedInstancesCount	30
password	33
pendingReplyCount	41
port	34
property	36
referenceUserContext	28
refreshOption	12
rows	15
selectionCriteria	7
serverAdapter	35
serverAdapterName	35
serverCheckOption	11
serverResponseTime	42
set<DataElementCode>Present(Boolean b)	96
subSchema	44
subSchemaList	43
tableModel	46
tableModel	45
updatedFolders	19
updatedFoldersTableModel	45
updatedInstances	20
updatedInstancesTableModel	46
userDetail	23
userId	32
userInputRows	21
userOutputRows	22
userServiceCode	24
userServiceCodes	24