IBM

VisualAge Pacbase 3.0

eBusiness Applications
Proxy Programming Interface

DDOVI000301A

First Edition (July 2002)

This edition applies to the following licensed program:

- VisualAge Pacbase Version 3.0

DDOVI000301A

NOTICES

TRADEMARKS

SUMMARY

A detailed Table of Contents follows this Summary.

Table of Contents

Foreword

CONTENTS OF THE MANUAL

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 15, describes the classes of the public interface, including inheritance trees.
- Chapter 2, *Attributes*, on page 19, 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 53, 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 Applications - Concepts & Architectures'* guide.

If you are new to this type of development, you may find it useful to read the '*eBusiness 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.

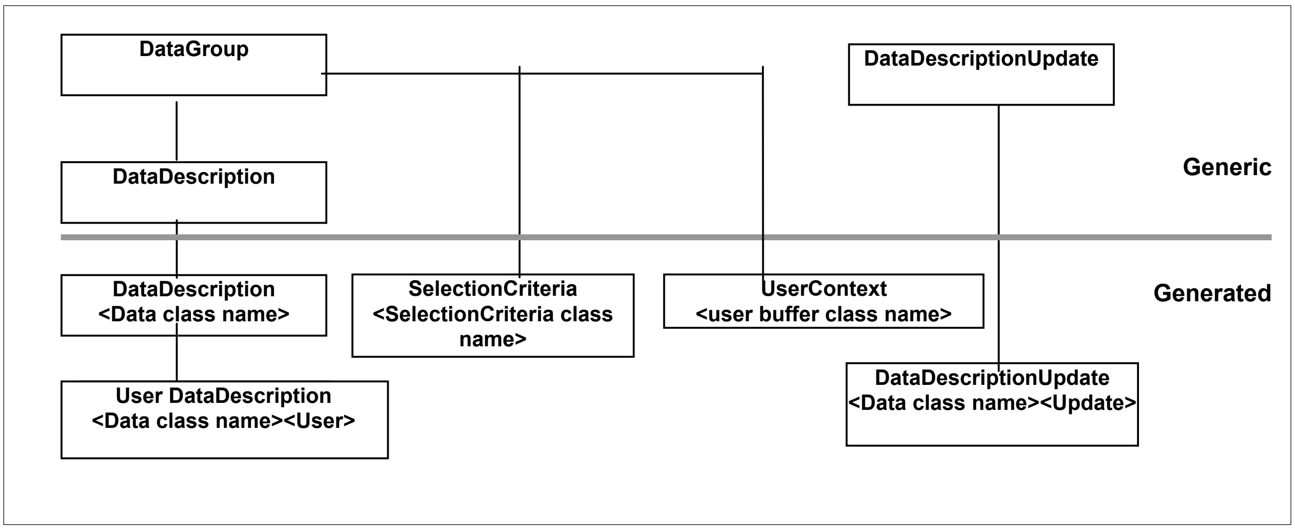
The indication: "Use Name" signals the corresponding label displayed in the Composition Editor, used in visual programming.

1. Classes

1.1. Data Classes

1.1.1. Inheritance Diagrams

1.1.1.1. Java and COM


DataGroup
DataDescriptionUpdate
DataDescription
Generic
Generated
DataDescription <Data class name>
SelectionCriteria <SelectionCriteria class name>
UserContext <user buffer class name>
User DataDescription <Data class name><User>
DataDescriptionUpdate <Data class name><Update>

. 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>

1.1.2. 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.

1.1.3. 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).

1.1.4. 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:

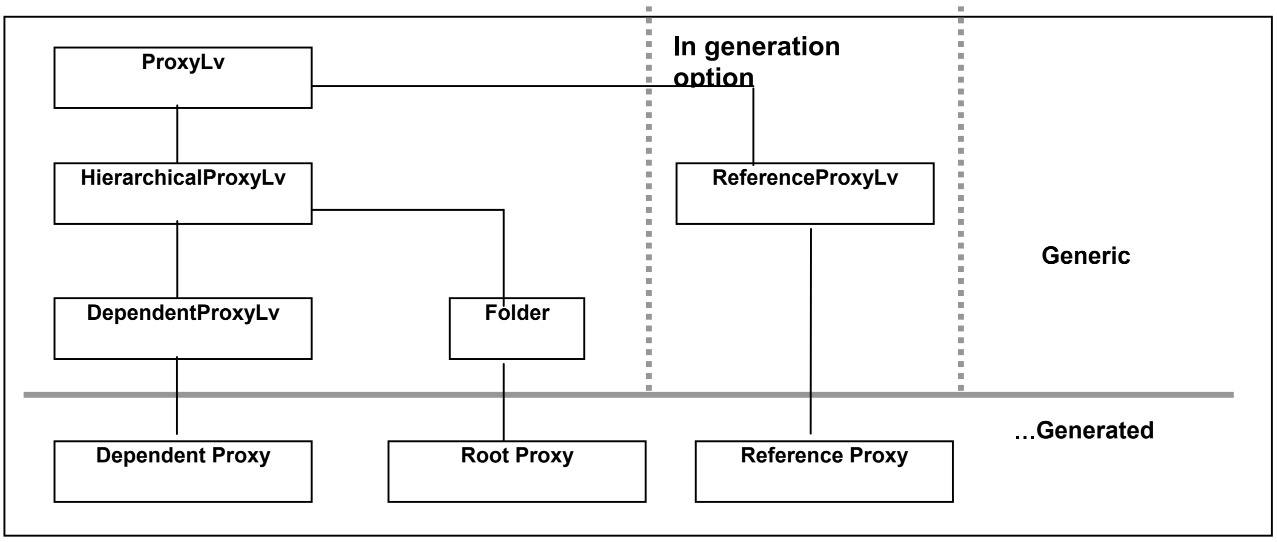
- **Created**: The Logical View associated with the node has been created locally.
- **Modified**: The Logical View associated with the node has been modified locally.
- **Deleted**: The Logical View associated with the node has been deleted locally.
- **Read**: At least one dependent instance in the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

1.1.5. 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.

1.2. ProxyLv Class: Inheritance Diagrams



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.

2.1. Management of Selections

2.1.1. 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`Use name `Selection Criteria`
• get/set C++	`public <LPDISPATCH> getSelectionCriteria()` / set not available

2.1.2. 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 attribut 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 Int getExtractMethodCodesCount()`
• Element	`public String getExtractMethodCodesElementAt(Int i)`

2.1.3. 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(StringExtractMethodCode)`

COM

• Type	`String`
• Internal code	`extractMethodCode`
• get/set C++	`public BSTR getExtractMethodCode()` / `public void setExtractMethodCode(LPCTSTR s)`

2.1.4. 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 isManualCollectionReset(Boolean aBoolean)`

COM

• Type	`Boolean`
• Internal code	`manualCollectionReset`

• get/set C++	`public BOOL getManualCollectionReset()`
	`public void setManualCollectionReset(BOOL b)`

2.2. Management of Updates

2.2.1. 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 aBoolean)`

COM	
• Type	`Boolean`
• Internal code	`serverCheckOption`

• get/set C++	`public BOOL getServerCheckOption()`
	`public void setServerCheckOption(BOOL b)`

2.2.2. 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 aBoolean)`

COM	
• Type	`Boolean`
• Internal code	`refreshOption`

• get/set C++	`public BOOL getRefreshOption()`
	`public void setRefreshOption(BOOL b)`

2.3. Exchange Flow Check

2.3.1. 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 max)`

COM

• Type	`Integer`
• Internal code	`maxNumberOfRequestedInstances`

• get/set C++	`public long getMaxNumberOfRequestedInstances()`
	`public void setMaxNumberOfRequestedInstances(long i)`

2.3.2. 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 globalSelection)`

COM

• Type	`Boolean`
• Internal code	`globalSelection`

• get/set C++	`public BOOL getGlobalSelection()`
	`public void setGlobalSelection(BOOL b)`

2.4. Logical View Instance Container

2.4.1. 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 attribut 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)`

2.4.2. 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 24) 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	`localSort`
• get/set	`public boolean getLocalSort()` / `public setLocalSort(boolean)`

COM

• Type	`Boolean`
• Internal code	`localSort`

• get/set	`public Boolean isLocalSort()`
	`public void setLocalSort(BOOL a)`

2.4.3. 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 24).
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	`com.ibm.vap.generic.Comparator`
• Internal code	`dataComparator`
• Use name	-

• get/set	`public com.ibm.vap.generic.Comparator getDataComparator()`
	`public void setDataComparator(com.ibm.vap.generic.Comparator c)`

COM

Not available.

2.4.4. 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 <Generated Type> detail()` / set not available

COM

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

2.4.5. 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

Available with a browsing API for collection-type attributes.

• Nb of elements	`public Long getUpdatedFoldersCount()`
• Element	`public {Name of generated class DataUpdate } getUpdatedFoldersElementAt(Long i)`

2.4.6. 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` de `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

Available with a browsing API for collection-type attributes

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

2.4.7. 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 24).

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)`

2.4.8. 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 24).

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)`

2.4.9. 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

2.5. Management of User Services

2.5.1. 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

Available with a browsing API for collection-type attributes.

• Nb of elements	`public Long getUserServiceCodesCount()`
• Element	`public String getUserServiceCodesElementAt(Long i)`

2.5.2. 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 code)`

COM

• Type	`String`
• Internal code	`userServiceCode`

• get/set C++	`public BSTR getUserServiceCode()`
	`public void setUserServiceCode(LPCTSTR s)`

2.6. Management of Logical Locks

2.6.1. 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

2.7. Management of Selection-Return Messages

2.7.1. 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

2.7.2. 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	`accesInfoKey`
• get/set C++	`public BSTR get AccessInfoKey` / set not available

2.8. Management of Events

2.8.1. 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()`

2.9. Management of Contextual Information

2.9.1. 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`
• Use name	`FolderUserContext`
• get/set C++	`public <LPDISPATCH> getFolderUserContext()` / set not available

2.9.2. 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}Buffer`
• Internal code	`referenceUserContext`
• get/set C++	`public <LPDISPATCH> getReferenceUserContext()` / set not available

2.10. Available Counters

2.10.1. 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

2.10.2. 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

2.10.3. 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 int 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

2.11. Management of Communications

2.11.1. 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 localisations 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)`

2.11.2. 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 (page 63).

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 newLoc)`

COM	
• Type	`String`
• Internal code	`location`

• get/set C++	`public BSTR getLocation()`
	`public void setLocation(LPCTSTR l)`

2.11.3. 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 uid)`

COM	
• Type	`String`
• Internal code	`userId`
• get/set C++	get not available / `public void setUserId(LPCTSTR u)`

2.11.4. 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 pwd)`

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

2.11.5. 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 host)`

COM	
• Type	`String`
• Internal code	`host`

• get/set C++	`public BSTR getHost()`
	`public void setHost(LPCTSTR h)`

2.11.6. 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 port)`

COM

• Type	Long
• Internal code	`port`

• get/set C++	`public Long getPort()`
	`public void setPort(Long p)`

2.11.7. 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

This information is not referenced as an attribute in the Proxy's API. It is possible to modify it by executing the following method:

• get/set	No get `public void setLocationsFile(String f)`

COM

• Type	`String`
• Internal code	`locationsFile`
• get/set C++	get non disponible / `public void setLocationsFile(LPCTSTR f)`

2.11.8. 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 adaptator.

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

This attribute is systematically available on the root node.

Java

• Type	`ServerAdapterName`
• Internal code	`serverAdapterName`
• Use name	`Server Adapter Name`

• get/set	`public String getServerAdapterName ()`
	`public void setServerAdapterName(String className)`

• Type	`ServerAdapter`
• Internal code	`serverAdapter`
• Use name	`Server Adapter`

• get/set	`public ServerAdapter getServerAdapter ()`
	`public void setServerAdapter(ServerAdapter serverAdapter)`

COM

• Type	`String`
• Internal code	`serverAdapterName`

• get/set	`public BSTR getServerAdapterName ()`
	`public void setServerAdapterName(LPCTSTR className)`

2.11.9. 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**
Folder and location	locationsFile	host
userId	traceFile	port
password	traceLevel	
connectionCleaningInterval	nbMaxConnection	
hostEncoding	connectionTimeout	
clientEncoding	codePageFile	

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` et `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 in 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 additionnal 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 additionnal 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.

2.12. Management of Asynchronous Conversations

2.12.1. Determining the Type of Conversation

Description

This attribute is a Boolean value defining the current conversation type of the Folder. It must de 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)`

2.12.2. 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 ServerActionContext getLastReplyContext()` set not available

2.12.3. 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	`<Foldername>ServerActionContext /`
• Internal code	`maximumReplyCount`
• get/set C++	`public LPDISPATCH getMaximumReplyCount() /` set not available

2.12.4. 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	`Integer`
• Internal code	`pendingReplyCount`
• get/set	`public short getPendingReplyCount()` set not available

COM	
• Type	`int`
• Internal code	`pendingReplyCount`
• Use name	`pendingReplyCount`
• get/set C++	`public int getPendingReplyCount()` set not available

2.13. Conversation Time

2.13.1. 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	`Long`
• Internal code	`communicationResponseTime`
• get/set	`public Long getCommunicationResponseTime()` set not available

COM

• Type	`Integer`
• Internal code	`communicationResponseTime`
• get/set C++	`public short getCommunicationResponseTime()` set not available

2.13.2. 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	`Long`
• Internal code	`serverResponseTime`
• get/set	`public Long getServerResponseTime()` set not available

COM

• Type	`int`
• Internal code	`serverResponseTime`
• Use name	`serverResponseTime`
• get/set C++	`public int getServerResponseTime()` set not available

2.14. Sub-Schema Management

2.14.1. 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 Int getSubSchemasCount()`
• Element	`public String getSubSchemasElementAt(Int i)`

2.14.2. 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)`

2.15. External Request Management

2.15.1. 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)`

2.16. Use of a JTable

2.16.1. 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 constitued 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

2.16.2. 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 constitued 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

2.16.3. 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 constitued 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

2.16.4. 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

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.

3.1. Actions Performed Locally

3.1.1. Updates

3.1.1.1. 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()`
• Use name	`createInstance`

3.1.1.2. 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()`

3.1.1.3. 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()`

3.1.2. Cancellation of Updates

3.1.2.1. 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 Updates`

COM

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

3.1.2.2. 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()`

3.1.2.3. Cancellation of Updates on a Node Intance

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 paramater.

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)`

3.1.2.4. 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()`

3.1.3. Management of User Services

3.1.3.1. 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()`

3.1.3.2. 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`

3.1.3.3. 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()`

3.1.4. 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.

3.1.4.1. 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}`

3.1.4.2. 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 getDetailFromRowsIndex(int index) throws LocalException`
• Use name	`getDetailFromRowsIndex`

COM	
• C++declaration	`public void getDetailFromRowsIndex (short index)`
• Internal code	`getDetailFromRowsIndex(Integer)`

3.1.4.3. 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)`

3.1.4.4. 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

Non disponible

3.1.5. Miscellaneous Initializations

3.1.5.1. 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()`

3.1.5.2. 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()`

3.1.5.3. 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()`

3.1.5.4. 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()`

3.1.5.5. 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()`

3.1.5.6. 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 resetSelectionCriterias()`
• Use name	`Reset Selection Criterias`

COM

• C++ declaration	`public void resetSelectionCriterias()`
• Internal code	`resetSelectionCriterias()`

3.1.5.7. 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	`initializeInstance`

COM

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

3.1.6. Management of Referenced Instances

3.1.6.1. 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 } 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)`

3.1.7. Retrieval of Proxies' Generation Contexts

3.1.7.1. 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)`

3.1.7.2. 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)`

3.1.8. Sub-Schema Management

3.1.8.1. 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()`
• Use name	`ResetSubSchema`

3.2. 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.

3.2.1. Selection on a Node

3.2.1.1. 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()`

3.2.1.2. 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`

3.2.1.3. Lecture d'instances à partir d'identifiants

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)`

3.2.2. Concurrent Selection on Multiple Nodes with or without Locking

3.2.2.1. 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()`

3.2.2.2. 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()`

3.2.2.3. 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()`

3.2.2.4. 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`

3.2.2.5. 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({}Data data) throws LocalException, ServerException, CommunicationError, SystemError`
• Use name	`Read First Children From {}Data`

COM

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

3.2.2.6. 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)`

3.2.3. Management of Paging

3.2.3.1. 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()`

3.2.3.2. 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' 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()`

3.2.4. Sending of Updates

3.2.4.1. 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()`

3.2.5. Management of Logical Locks

3.2.5.1. 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()`

3.2.5.2. 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`

3.2.6. Management of Dependent Instances

3.2.6.1. 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()`

3.2.7. Management of User Services

3.2.7.1. 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()`

3.2.8. Management of Asynchronous Conversations

3.2.8.1. 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)`
• Code interne	Boolean getReply `(<Foldername>ServerAction)`

3.2.8.2. 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.Server ActionContext aContext)`
• Use name	`Checks the validity of the request reply`

COM

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

3.2.9. 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`

3.2.10. 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()`

3.3. 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.

3.3.1. 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	`createRequest`

COM

• C++ Declaration	`public void createRequest()`
• Internal code	`createRequest()`

3.3.2. 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	`sendRequest`

COM

• C++ Declaration	`public void sendRequest()`
• Internal code	`sendRequest()`

3.3.3. 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()`

4. Events

4.1. 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 ‘).

4.1.1. 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`

4.1.2. 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`

4.1.3. 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`

4.1.4. 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`

4.2. Management of Unit Reads

4.2.1. 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`

4.3. Management of Simultaneous Selections

4.3.1. 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`

4.4. Management of Logical Locks

4.4.1. 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`

4.4.2. 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`

4.5. Management of Dependent Instances

4.5.1. 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`

4.5.2. 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`

5. Public Interface for Data Elements Handling

5.1. 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> (Type t )`

5.2. 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)`

5.3. 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

Available with a browsing API for collection-type attributes.

• Nb of elements	`public Long <DataElementCode>ValidLabelsCount()`
• Element	`public Type <DataElementCode>ValidLabelsAt(Long i)`

5.4. 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<CodeRubrique>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.

5.5. 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..

5.6. 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	`initializeFrom`

COM

Non disponible.

5.7. Management of a Data Element's Presence

Description

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

The first action is always available for Data Elements defined in the `DataDescription` and `UserDataDescription` classes

The second action is 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 (Boolean b)`

• Internal code	Boolean `is<CodeRubrique>Present/setCodeRubriquePresent(Boolean value)`

5.8. 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(L          ong, Boolean)`

5.9. Management of membership to a Sub-schema

5.9.1.1. 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 indexRubrique)`

• User name	`Belongs to current subschema`
COM	
• C++ declaration	`public BOOL belongsToSubschema(short index)`
• Internal code	`Boolean belongsToSubSchema(Integer)`

5.10. 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.

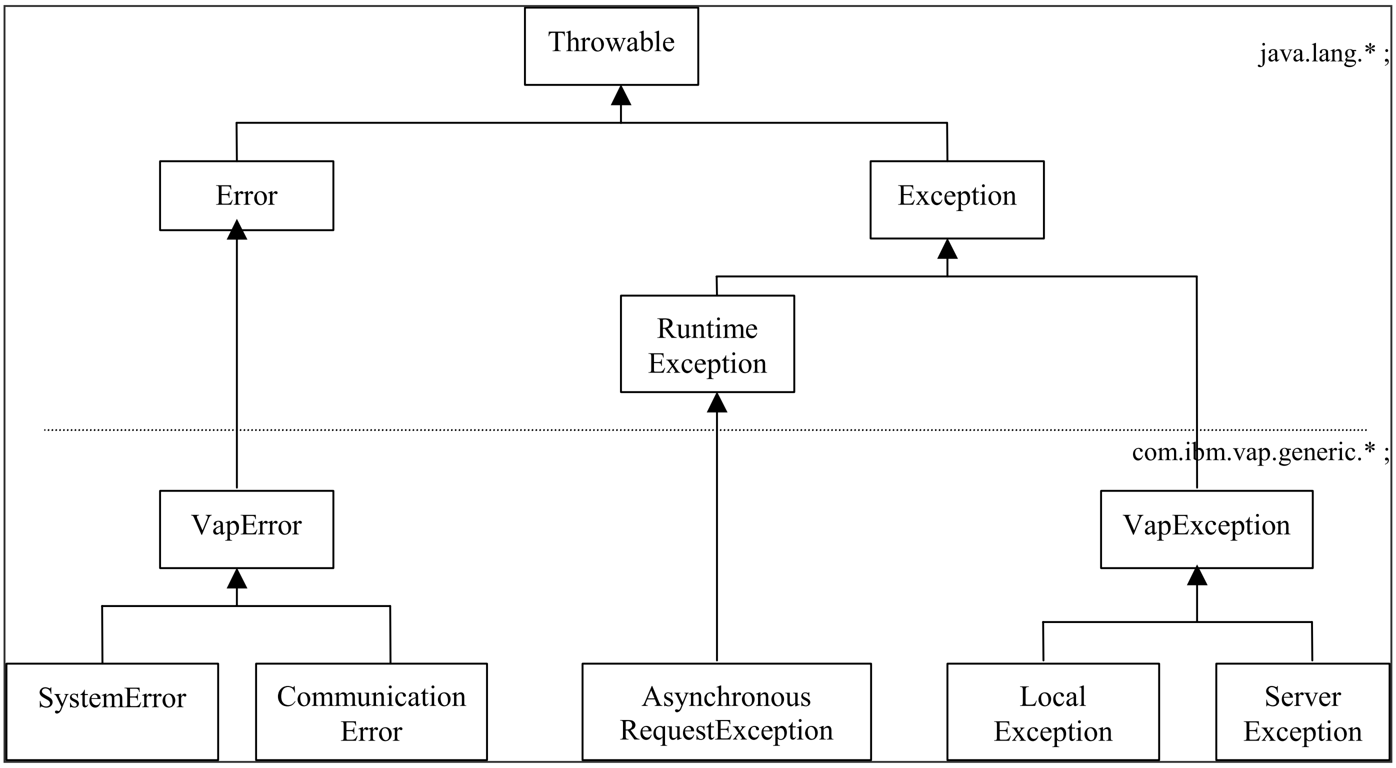
Java	
• Declaration	`public boolean dataFieldBelongsToExtractMethod (String FieldCode, String extractMethodCode)`
• Internal code	`DataFieldsBelongsToExtractMethod`
COM	
• C++ declaration	`public BOOL dataFieldBelongsToExtractMethod (LPCTSTR FieldCode, LPCTSTR extractMethodCode)`
• Internal code	`Boolean dataFieldBelongsToExtractMethod(String, String))`

6. Management of Errors

6.1. 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 developper 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).

6.1.1. Classes Related to the Management of Errors

6.1.1.1. 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).

6.1.1.2. 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 Applications - Graphic Presentation*' guide for the list of the system errors.

6.1.1.3. 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 Applications - Graphic Presentation*' guide and also in the HTML documentation associated with the generic classes: Package `com.ibm.vap.generic.`

6.1.1.4. 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()`).

6.1.1.4.1. 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 6.1.2, *Customizing Error Messages*, on page 102.

6.1.1.4.2. 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.

6.1.2. 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 technics of labels provided by Java language: the differents 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 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()`).

6.1.2.1. 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.*

6.1.2.2. 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 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.

6.1.2.3. 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.

6.1.2.4. 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_REFERING_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
```

6.2. 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.

6.2.1. 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`

6.2.2. VapError Attributes

6.2.2.1. 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()`

6.2.2.2. 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 ()`

6.2.2.3. Management of the Error Key

Operation

This action enables to retrieve the error key (refer to the *'eBusiness Applications – Graphic Presentation* ' guide for the list of errors).

• SignatureC++	`public BSTR getType()`
• Code interne	`String getKey`

6.2.2.4. Management of the Error Label

Operation

This action enables to retrieve the default or customized error label (cf. section 6.2.4 *Customizing Error Messages* for information on the customization procedure).

• C++ declaration	`public BSTR getLabel()`
• Internal code	`String getLabel()`

6.2.2.5. 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`

6.2.3. 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.

6.2.3.1. 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`

6.2.3.2. 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`

6.2.3.3. 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`

6.2.3.4. 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`

6.2.3.5. 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`

6.2.4. 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.

6.2.4.1. Naming Rules for Error Messages Files

The following rules apply to the storage of error messages:

- As much VapErrorMsg text files as targetted 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.

6.2.4.2. Syntax of Error Messages Files

To modify or create a file of error messages, a number of syntaxic rules must be followed Any syntaxic error triggers the complete invalidity of the externalization of error messages.

- Use quotes to delimite 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.

6.2.4.3. 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`****.*

6.2.4.4. 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 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.

6.2.4.5. 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.

6.2.4.6. 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_REFERING_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_EXIST = 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.
```

7. Index

Java events

Java methods

Java properties