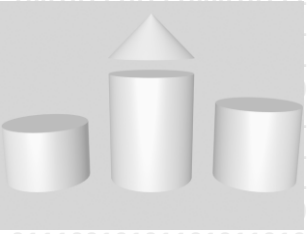


Reinhard Karge

Active Data Link 2.0

December 2002

January 2007 (revised)

run Software-Werkstatt GmbH
Weigandufer 45
12059 Berlin

Tel: +49 (30) 609 853 44
e-mail: run@run-software.com
web: www.run-software.com

Berlin, October 2012

Content

1 Introduction 4
- Active Data Link 4
- (ADL) 4
- Scope 4
- Events 5
- Action control 6
- Summery 7

2 ADL Overview 8
- Database Management System (DBMS) 8
- GUI Interface 9
- Active Data Link 9

3 Data Access Handle 11
- Cursor 11
- Access Hierarchies 11
- Subsets 11
- Data source 12
- Data states 13
- Translating data states 14
- Data Events 14
- Event caching 16
- Further requirements 17
- Databases 18
- Summary 19

4 Data Control Handle 20
- Control Handle Types 20
- Links to Data Access Handle 21
- Control states 21
- Control events 23
- Summary 24

5 Application Events ..... Error! Bookmark not defined.

6 Practical experience ..... Error! Bookmark not defined.
- The paper ..... **Error! Bookmark not defined.**
- Development ..... **Error! Bookmark not defined.**
- Current state ..... **Error! Bookmark not defined.**

1 Introduction

Active Data Link is a technology, which has been developed by run-Software from 2003-2004. The idea is basically to reduce the costs for application development by 80 or 90%.

Considering, that application development mainly deals with accessing and moving data from and into forms to display or receive the current data, this process can be automated nearly by 100%.

Relational development environments have done the first step in this direction by displaying tables or views associated with a GUI element (usually a table grid). Application logic is, however, a bit more complex. In many cases, applications follow a hierarchical procedure when accessing data or are at least mainly controlled by data selection. In many cases, actions provided within an application do not depend on the application but are defined as business rules, which is not part of the application logic.

Active Data Link (ADL)

ADL describes a technology, which links the application tightly to the data access. Thus, the application is driven by data access, but data access is driven also by the application reacting on application events.

Scope

The intension of ADL is to support large and complex applications. ADL is described here for running on application clients. In principle it is also possible to run ADL on an application server, supposed that the application server generates the GUI events required by ADL.

DBMS

ADL is not based on a particular database management system (DBMS). Practically one may implement ADL for any type of DBMS. It requires, however, more resources implementing ADL on a relational DBMS than on an object-oriented DBMS.

Data access mechanisms differ, however, from relational DBMS or .NET access mechanisms, which both are table oriented, while ADL requires instance and path oriented access.

GUI

Most GUI systems (as Visual Basic– Microsoft, Delphi – Borland, or QT – Trolltech) can refer to as base for an ADL system. The only requirement for the GUI system is, that it passes a minimum of GUI events to the application and allows creating GUI elements on the fly.

Events

ADL is based on event generated on the database side as well as on the GUI side.

Data Events

There are a few typical events, which cause the application to react as selecting a line in a list or passing the focus to a certain GUI-Element. This action causes a data read action, which generates a number of read events that automatically lead to update the data displayed on the user interface. This is just a rough explanation of the mechanism behind ADL.

But “record access” does not generate sufficient events for driving an application. Selecting a person, the application might need to display the person’s children in a subsequent list control. The set of displayed children probably changes always when selecting a new person. Hence, ADL requires something like a collection changed event. But also collection update events, which indicate that a collection may have changed, must be supported to update e.g. the list of children displayed for a person.

Later on in this paper I will summarize the most important events, which should be supported by an ADL system.

GUI Events

Considering a simple application, which displays a list of persons in a list and the details for the selected person in a form right of the list (which is typical for many applications).

When selecting a person with a mouse, the user expects the person data being displayed in the right form. With ADL, the application would react as follows:

- The click event on the list (current list item changed) causes ADL to read the data record for the selected instance from the associated collection.
- The read event for the data record also generates read events for each property, which cause filling in data into GUI controls linked with the data items (properties).

This mechanism sounds very simple, but it will reduce the development costs extremely.

Action control

Most applications need to activate several actions, which often are considered as application rules. In many cases, it turns out, however, that actions are rather based on business rules than on application rules. Any time, the question whether an action makes sense outside the application context or not, is answered with yes, the action is probably a business rule and had been implemented as application rule just for practical reasons.

Action elements

Typically, actions are associated with action elements that can be activated by the user (buttons or menu items) or with GUI events. This is the same in an ADL environment, i.e. ADL does not add anything new to this aspect, except that actions may cause data events, which might influence the application again.

On the other side application actions can be used for generating data events in order to control the application. E.g. selecting a data record in a collection may cause selecting an instance in the user interface and mark is as selected or current instance. Driving an application by accessing data provides a number of valuable features, which might be used designing an application.

Business rules

As well as application rules, business rules may influence the application behaviour in an ADL environment. This is, sometimes a positive effect, but sometimes, it might be disturbing. Hence ADL must provide a feature for suppressing data events.

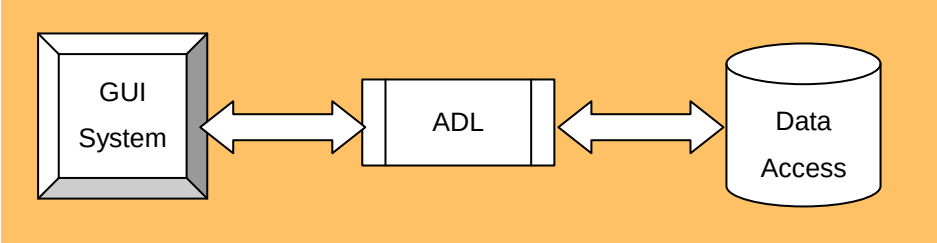
Summery

ADL is just an extension for event driven applications. In contrast to traditional event driven GUI applications ADL adds data events to application control. This paper describes the pre-conditions ant rules for setting up an ADL environment.

In principle, any one may design its own ADL environment, but it takes some time to get it running properly. ADL mainly requires a data access interface, which is able to fire different classes of application events and a proper GUI interface, which fires GUI events.

2 ADL Overview

The main task of ADL is translating data events into GUI actions and GUI events into data access actions.



In principle, ADL can be provided for any kind of data base, but it requires more resources implementing ADL for a relational DBMS than for XML or an object-oriented DBMS.

Database Management System (DBMS)

The reason is simply, that relational DBMS are table oriented while XML and object-oriented databases support instance (element) access and paths. Object oriented databases provide also some additional prerequisites, which are very helpful when designing an ADL environment.

Data Access Handle

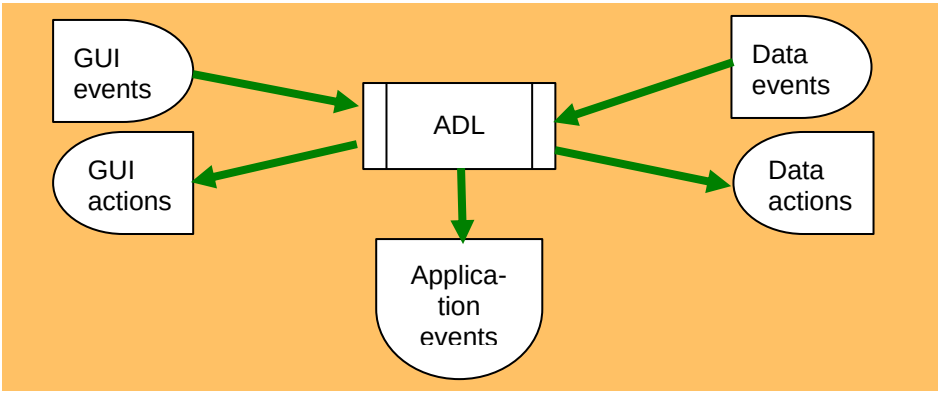
A simple way to support ADL's data side is providing a class, which is called here Data Access Handle. A data access handle. A data access handle provides the functionality for handling data in a simple (atomic) data field like number or text but also for handling complex data as collections or hierarchical structures.

To provide a common data interface, data access handle must also support transient and persistent data access.

Data Events

The data access handle generates relevant data events, which are not part of the data model or functional application model. Data events are generic events which indicate that data has been read or updates. Such events cannot be triggered by database triggers, since the database model should not depend on the application model.

Unique identifier	ADL becomes very difficult, when database instances (records) do not have a unique identifier. This makes it nearly impossible to synchronize the application state with the data access manager state. When the database system does not support unique identifiers, the data access handler of ADL should do this.
Collections	XML and object-oriented databases usually support local collections for associations and relationships. Since applications usually work hierarchically, local collections and their state are an important information or ADL. Hence, the data access handle must provide information about any related collection, as collection identifier, which uniquely identifies a collection or cardinality.
Modification	Data access handler must inform the application about any sort of modification on a field, instance or collection. Since modifications might be caused by other applications, which not necessarily generate modification events, the data access handler must be able to detect data modifications whenever required.
GUI Interface	The GUI-side of ADL is represented by class that is called here Data Control Handle. As well as the data access handle, the data control handle refers to any kind of GUI element, which can be linked to data. In this sense, a data control handle may support an elementary data field (number or text) but also a form displaying parts of an instance or a table or tree structure.
Data control handle	Data control handle provide functionality for controlling the GUI element behaviour (setting size and colour, filling in data etc). The data control passes the ADL requests to the underlying GUI elements, which might VB, Delphi or QT elements.
GUI events	Practically, one may connect data control handle to any GUI system, which generates the GUI events required by ADL for controlling the application. The number of events requested by ADL is, however, limited and supported by most of GUI systems.
Active Data Link	Active data link provides the connection between Data Control Handle and Data Access Handle. This includes a mechanism for translating GUI events into data actions and reverse (data events into GUI actions).



ADL itself passes GUI events to the application that may also react on specific events. On the other hand, the application may submit GUI or data requests via ADL.

3 Data Access Handle

The data access handle supports data actions (e.g. locating a data instance) and generates required database events. ADL allows linking a data access handle to any number of data control handles (GUI side). When generating data access events, resulting requirements are sent to all data control handles linked to the data access handle.

A data access handle may refer to an elementary data field, a data instance (or record) or to a data collection. Since a data instance may refer to a number of collections, again, a data access handle may also refer to a number of collections or a number of collections of collections etc. Thus, the complexity of data handles by a data access handle is practically unlimited.

Cursor

Data access handles provide cursor functionality, which allow selecting one instance in a collection as “current” or “selected” instance.

Database handles for elementary data usually contain exactly one instance, which by default is considered as selected.

Access Hierarchies

Data access handle may form hierarchies, i.e. for each property of an instance for the data access handle a subordinated data access handle can be created.

```
DAH  person(database,”Person”);
DAH    p_name(person,”first_name”);
DAH    children(person,”children”);
DAH      c_name(children,”first_name”);
DAH      grand_children(children,” children”);
```

Subsets

In many cases applications display subsets of a relational table, i.e. children as a subset of persons. In object-oriented environment children becomes a property of person, which contains the persons that are children of the selected person.

A subset defined in a subordinated data access handle may change, when the selected instance for its parent changes (e.g. changing the selected person will result in another children subset in the children access handle.

In a relational environment this information is available as well, but must be calculated from the attributes in the person table or from a many-to-many relationship table.

For derived sub sets it becomes more difficult to provide unique identifiers for identifying a sub set, which are strongly recommended to make ADL performing properly.

Another extension is very useful, but not necessarily required for ADL. This is the definition of base collections. When working with local collection, in many cases on collection can be defined as superset of another collection, e.g. Persons as superset for person's children (absolute super set) or employees of a company as super set for the head of a section in this company (relative super set).

As well as enumerated values, defining supersets is a simple way of automatically displaying choice values in a drop combo box or in a radio button control.

Data source

The data handle by a data access handle might be defined in a number of different ways.

Constant

Constant data is a way to define data for headlines in a multiple region list or for other purposes. Constant data access handles are positioned after being opened and will never change the state (even though they may change the value).

Transient data

An access handle may refer to transient data, which is not stored externally and available within the application, only. Transient data can be updated by the application. In many cases transient data contains derived values as sums or percentages.

Global collection

Data access handles for global collection refer to extents (OODBMS) or tables (RDBMS).

Property

Data access handles for properties refer to attributes or local collections defined for the instances in the parent data access handle.

Expression

Data access handles for expressions (SQL or OQL queries or access paths) may refer to elementary data, instances or collections calculated according to the expression or access path.

Since OQL queries or access paths can be defined relative to the selected instance, the will perform better than SQL queries, because detecting dependencies for an SQL query is more difficult.

Data states

Data states describe the state of a data access handle, i.e. whether it is opened, an instance is selected etc. Data states do not depend on the type of data source handled by the access handle.

The state of a subordinated data access handle depends on its parent. In the example above this means, that the data access handles for children and p_name are invalid as long as no instance in the data access handle for person is positioned.

Thus, data for subordinated data access handles can be accessed (located, updated etc.) only, when its parent is selected and positioned. Unselecting the parent automatically invalidates all children.

Most relevant data access handle states are:

Invalid

A data access handle is opened, when it not is valid but has been successfully creates, e.g. when it has a parent and the parent is not selected.

Opened

A data access handle is opened, when it not is valid but has been successfully creates, e.g. when it has a parent and the parent is not selected.

Valid

The data access handle is valid, i.e. one may select or position an instance. A data access handle is valid, when id does not have a parent and is opened or when its parent has the state Selected.

(implies opened)

Selected

An instance in a collection has been selected (but not necessarily read).

(implies valid)

Initialized	A new instance in a data access handle has been provided. All subordinated database handle for attributes are considered as Initialized as well. (implies valid)
Positioned	An instance in a data access handle has been selected and read. In this state all subsequent data access handles are valid. Data access handles for attributes (elementary or complex) are automatically positioned, when its parent is positioned. (implies selected)
ReadOnly	An instance in a collection is positioned but cannot be updated. (implies positioned)

Translating data states

Data states can directly be translated into essential data control handle states. Data control handle states are described in “Data Control Handle”.

Data states area directly translates into control states as follows:

Data state		**Control State**
ReadOnly	→	ReadOnly
Positioned	→	CanUpdate
Initialized	→	CanUpdate
Selected	→	CanCreate
Valid	→	CanCreate
Opened	→	Disabled
Invalid	→	Inactive

Data Events

Data events are fired, when the data access handle changes. Whenever a data event happens it is sent to the linked data control handles calling an appropriate control action.

Data events can be divided into two sub-groups: Property events and instance events.

property opened
: A data access handle has been opened. This event causes the data control handle to activate itself. This may also include positioning the data access handle, which, again, will generate a read event.

property reset
: A property reset event is generated, when the data access handle state turns into opened (e.g. from valid or positioned). This is usually the case, when the parent for a data access handle changes the positioned instance.

  The property reset event causes the control handle to clear all data displayed.

instance reset
: An instance reset event is generated, when the data access handle state turns from positioned or selected into valid. This is usually the case, when changing the selection instance in the access handle.

  The reset event creates property reset events for all subordinated data access handles.

instance deleted
: The deleted or removed event results in a *valid* state for the data access handle. This event is generated instead of an instance reset event in this case, to indicate that the instance has been reset because of a remove or delete operation for the currently selected instance.

instance located
: A located event is fired when the access handle state changes to selected or when the selection changes. Changing the selection in the access handle first changes the access handle state into valid before it is changed to located.

instance read
: A read event is fired when the access handle state changes to positioned. This also happens when the positioned instance for a data access handle is changed.

  The instance read handle generates property read events for all subordinated data access handles.

  The read event might be preceded by a reset event in case of changing the positioned instance, since this will be unselected before positioning the new instance. Moreover, a locate event is generated, since the instance to be positioned must be located before.

instance inserted
: The inserted event results in a *positioned* state for the data access handle. This event is generated instead of an instance read event in this case, to indicate that the positioned instance has been created before being read.

instance updated
: The updated event does not change the *positioned* state for the data access handle. This event is generated to indicate that the positioned instance has been updated.

property updated
: The property updated event indicates, that an elementary value has been updated or that the instances referenced in a collection (global or local set) have been changed, which is usually the case, when instances have been added to or removed from the collection or when the currently active index for accessing the instances has been changed.

property refresh
: A data access handle, which had the state *opened* changed to state *valid*. This usually happens for subordinated data access handles, when the parent handle changes its state to *positioned*.

  Note, that changing an instance in the data access handle first unselects the current instance, which changes the data stated to *opened* for all subordinated data access handles and generates a reset events. Afterwards, positioning the new instance will change the state again to *valid*, which generates a refresh event.

  A refresh event causes the control handle to initialize its data. In case of collection controls, this usually results in filling in the list.

instance initialized
: Changing the state to *initialized* will cause a fill data action, which fills in initial data into the controls.

Event caching
: The list of generated events is not complete but describes the most relevant events in the context of ADL. Since any simple action may generate a number of read events, an event cache mechanism becomes necessary.

  Event caching not only collects events fired while performing a data access handle operation but also optimizes events by overwriting low priority events. This becomes even more important when considering the fact, that business rules may cause additional data access handle events.

  Events are optimized for each data access handle separately.

Implicit caching

Implicit event caching is done automatically while running a data access handle action.

In case of calling an `eraseCollection` action, the data access handle will fire an instance reset event at the end of the action even though a number of instances has been removed. Moreover, all subordinated access handle will fire one reset property event.

Application event caching

Events can be cached in the application starting and stopping event caching. This is necessary to allow complex operation in the application, but being able to synchronize the GUI state after the action has been completed.

Disable events

Another feature required is disabling events. This becomes necessary to perform internal or application actions, which should not be reflected on the user interface.

When disabling events, the application is responsible to synchronize the data access handle state with the current GUI state.

Inactive data access handles

In some cases generated read events may heavily disturb the application behaviour, especially when handling asynchronous data control events. To avoid this, inactive data access handles, which do not generate events, are required.

Further requirements

Besides supporting data navigation and event generating, data access handles must provide two more essential features to act successfully in an ADL environment.

One problem ADL creates are recursively working events, i.e. a read event creates read events or refresh events for all subordinated access handles which again may cause further read events. On the other hand, especially read events are very often generated and should not cause any action, when the control state does not change, i.e. when the instance or collection is still the same and has not been modified.

Identity

Since there is no direct connection between data access handle and data control handle, ADL becomes very inefficient when instances cannot be identified. Thus, the access handle must provide unique identities for selected instances as well as for current collections.

Modification count	An application cannot trust that all updates and changes are signalled properly to the application. Other applications may change data without signalling this to ADL or the application. Hence, data displayed in the controls might become outdated. To ensure, that data is always up-to-date, ADL requires a modification count for instances and collections (including subsets), which allows detecting external modifications on instances and collections.
Databases	ADL can be built in principle based on any type of database or data source. As long as identities and modification counts are not supported, ADL requires an ADL data source layer, that manages identity and modification count for the underlying data.
Required information	Since XML or relational data bases usually do not maintain modification count and instance identity, additional implementation effort is required for running an ADL implementation based on relational databases.
Events	In order to provide an ADL based system, generic data events have to be supported by the used database management system. Generic data events as updated or deleted are supported by most database systems, but normally on instance level, only. In order to implement ADL, also generic collection events are required (e.g. inserted, removed).
Event handler	Many relational databases support triggers for handling events, but those are implemented for specific tables, i.e. one cannot refer to generic event handlers. Moreover, relational databases provide data usually as result of queries, e.g. in record sets or comparable constructs. Generic event handlers, which may register to access handlers as record sets (MS SQL Server) or property handles (ODABA) are necessary for providing ADL for a database system.

Which DBMS to use

Any database system, which supports generic data events (and most database system do so) for both, instances and collections (not so many support collection events) is a candidate for implementing ADL. Moreover, the database system has to support generic data event handlers, which is also not a very common feature.

One example for an ADL implementation is the ODABA GUI framework that also supports ADL for relational databases by adding an independent ADL database layer, which allows running ADL for most relational databases.

It depends on the further development of big database providers, whether such an development becomes necessary or not.

Summary

In an ADL environment the data access handle is tightly connected with the data control handle.

- Event mechanisms indicating state transitions in the access handle are one possible way of connecting access handles with control handles.
- Event caching helps to avoid an event inflation and allows optimizing generated events
- Disabling events is a risk since ADL cannot guarantee consistency between the access handle state and the control states. Nevertheless, ADL becomes difficult to handle without such a feature.
- For handling asynchronous control events, inactive data access handles are required.
- Identifier for instances and collections (including subsets) are required for optimisation.
- Modification counts for instances and collections are required for guaranteeing consistency.

4 Data Control Handle

Data control handle are used to display data in different ways on the user interface. Data displayed in control handles depends on the application state and the data source providing the data.

Data sources for control handles might deliver persistent or transient data. Since in ADL the control handle communicates with a data access handle, this will not make any difference.

Control Handle Types

Control handle are of different complexity. Control handle specialisations (specialised control handle types) may react differently on events generated by data access handles.

Control handle types can be classified into four relevant groups:

static

Static control handles are not linked to a data access handle and display an uncontrolled data value (usually a constant text or image).

elementary

Elementary control handles display elementary data. Usually, but not necessarily, elementary control handles are linked with an elementary data access handle, which provides elementary data (as number or text).

DropComboBoxes are considered as elementary, since they refer to elementary data, which is displayed in the line field on top. The droppable list, however, is a collection control, which usually is linked to the base collection (super set) of the displayed instance.

collection

Collection handles display collections of data, i.e. usually global (table) or local sets (subsets). Typical collection type controls are lists and table grids. Usually collection control handles display instances in a collection in a number of lines (one line for each instance).

complex

Complex control handles usually consist of several subordinated control handles, which may display properties of one or more instances. Typical complex control handles are forms and sub-forms.

Links to Data Access Handle

Each control handle (except static) is usually linked with exactly one data access handle. Elementary and complex control handles refer to the positioned instance in the data access handle. No data is displayed for those controls, when no instance is positioned in the data access handle and when no instance is initialized.

Collection control handles in most cases refer to a global collection (table) or are linked to a local collection (subset) by a subordinated data access handle.

Multiple data sources

Collection controls can be linked, however, to more than one data source for displaying more than one collection in the same list (e.g. parents and children). It depends on the GUI system, how those multiple collections are represented in the control.

ADL supports a hierarchical structure for regions and columns.

Regions

ADL data control handles support any number of regions in a data control. Each region is linked to a data access handle.

Each region may have any number of sub-regions, regardless whether the control is a tree or a table. Moreover, regions can be defined recursively:

```
Region  persRegion(person);
Region  childRegion(persRegion,"children");
Region  childRecursive(childRegion);
```

Region structures might be very complex and it is the task of ADL to optimize displayed data in large trees or lists.

Columns

Each regions may display a default key as text in the list or may define a number of columns, where each column is linked to a subordinated data access handle of the region access handle.

When a column refers to complex data, the column may have sub columns, which is linked again to subordinated data access handles for the column handle.

Control states

Data control handles do have a number of relevant states, which control the behaviour of the control.

Data state	The data state in a control is stored mainly as unique identifier and last mod count. Those information are sufficient to determine, whether data in the display need to be updated or not. Data states are stored based on the displayed information, i.e. not per data access handle linked to the control but per displayed data item. That means that in a table or tree each data cell has its identity and modification count information. If not, ADL will not behave consistently.
Has focus	The has-focus toggle state indicates, whether the control has the focus or not. Usually, actions related to the control can be activated only, when the control has the focus.
Hidden	The hidden or visible toggle state indicates, whether the control is visible to the user or not. When not being visible, the control will not update displayed data when receiving an update or read event. When changing the state to visible (or not hidden), the data displayed in the control will be updated according to the instance identifier and modification count.
Disabled	The enabled or disabled toggle state indicates, whether the control behaves or not. When being disabled, the control will not react on any user input including displaying context menu or reacting on function keys. Usually, disabled controls do not even get the focus.
Read only	The read-only or update toggle state indicates, whether data can be entered in the control. For multiple data source controls the read-only state is set for each data cell (region/column) displayed in the control. Read-only state still allows context menus and supports function keys. Some of the default actions are disabled.
Properties	Even though a number of other control properties as size, colour or position, determine the state for the data control, those are not relevant for ADL. Nevertheless, functionality is required for the data control handle to support modifications of these properties.

Control events

Control events are fired to indicate a state transition for the control, but also to submit asynchronous requirements. The list of handled events is not complete but considers the most relevant and typical GUI events and the way they are handles in ADL.

get focus

When the control gets the focus, the data access handle should be checked for modification. In case of collection handles the visible collections are checked for modification and updated when required as well as the current or selected instance in the list.

In case of modifications data in the control is re-read and updated. Reading may cause several events for subordinated data control handles linked to subordinated data access handles.

loose focus

When a control looses the focus, it stores data to the database, when data has been updated. This happens also, when a cell has been edited (in which case it temporarily becomes a subordinated control handle to the list) and leaves the edit state. Storing data in the data access handle will cause an instance update event.

current changed

The current changed event is fired from collection control, when the currently selected instance has been changed by selecting another one. This event causes the data access handler linked to the current region to locate (or read) the newly selected instance.

It will also check its sub regions for updates and refill, when required.

In case the region has a hierarchy of parent regions, for each parent region the data item will be positioned.

Thus, changing the current list item may cause a number of read and locate actions, which often have a strong influence on the data displayed in the application.

paint

The paint event is an asynchrony event generated by the control handle to update displayed data. For updating displayed data a number of read actions are required which may heavily disturb the appearance of the application.

To avoid this, collection controls create an inactive copy of the linked data access handle, which is only used for asynchronous updates and which does not generate data events.

Summary

To get ADL properly running, the data control handle interface must fulfil the following requirements:

- All investigated GUI systems (QT, Visual Basic, and Delphi) support the minimum requirements for an ADL system and can be used as GUI system for ADL.
- Data control handle must support elementary, complex and collection controls. Moreover, complex applications require multiple data source controls.
- The control states described in this chapter must be supported. This, however, is the case in most GUI systems.
- The GUI system must fire events comparable to the control events listed above. Usually, GUI systems generate much more events, which ADL can react on, but the listed events are the essential ones.