OpenEdge® Web Paper: Application Development Environment (ADE) Addenda
# Table of Contents

## Preface

- Purpose ............................................................................................................7
- Using ABL documentation ................................................................................7
- References to ABL compiler and run-time features ........................................8
- References to ABL data types .......................................................................8
- Typographical conventions .........................................................................8
- Example procedures ....................................................................................9
- OpenEdge messages ..................................................................................10
- Obtaining more information about OpenEdge messages .............................11

## Chapter 1: ADM Addenda

- Behavior change affecting procedure handle availability ..................................14
- New DynCombo properties .........................................................................14
- New DataView class ..................................................................................14
  - Changes to the ADM2 hierarchy ...............................................................15
  - Query property changes ......................................................................16
- Enabling columns for new records only ......................................................17
- Foreign values always applied when OpenOnInit is off ..............................17
- Container publishes "createObjects" from createObjects ..............................18
- Query manipulation on cached DataObjects ..............................................18
- Optimistic lock works for deletions ...........................................................19
- NO-LOCK and read-only SDO tables ...........................................................19
- EnableRule changes ................................................................................19
  - Find and filter actions EnableRule extended to disable when no parent. ....19
  - New canUpdate() function for tableio and browsetoolbar actions ..........20
- SDO always retrieves IndexInformation on first request ..........................20

## Chapter 2: Progress Dynamics Addenda

- Generated toolbars and translations ............................................................22
- New log file for Deploy Static Objects tool ................................................22
- Decimal values in dynamic combo-boxes ....................................................22
- New 4GL Generator options .....................................................................23
- Objects without deployment types ..............................................................23
- Context help for SmartDataFields ..............................................................24
- General Manager API behavior change ......................................................24
- Interrupting triggers on dynamic viewers ................................................24
- Translating Filter/Find windows .................................................................25
  - Where to translate items ....................................................................25
Preface

For details, see the following topics:

• Purpose
• Using ABL documentation
• Typographical conventions
• Example procedures
• OpenEdge messages

Purpose

This Web paper collects notes and enhancements about the application development environment from Progress® OpenEdge® Release 10.1 onward. It covers the following topics:

• ADM Addenda on page 13
• Progress Dynamics Addenda on page 21

Using ABL documentation

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is ABL (Advanced Business Language). With few exceptions, all keywords of the language appear in all UPPERCASE, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.
For the latest documentation updates see the OpenEdge Product Documentation Overview page on Progress Communities:


References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the ABL Virtual Machine (AVM). When the documentation refers to ABL source code compilation, it specifies ABL or the compiler as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies the AVM as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler's allowance for parameter passing and the AVM's possible response to that parameter passing at run time: "ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error." The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: "The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets."

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all UPPERCASE, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever integer appears, this is a reference to the INTEGER or INT64 data type.
- Wherever character appears, this is a reference to the CHARACTER, LONGCHAR, or CLOB data type.
- Wherever decimal appears, this is a reference to the DECIMAL data type.
- Wherever numeric appears, this is a reference to the INTEGER, INT64, or DECIMAL data type.

References to built-in class data types appear in mixed case with initial caps, for example, Progress.Lang.Object. References to user-defined class data types appear in mixed case, as specified for a given application example.

Typographical conventions

This documentation uses the following typographical and syntax conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.</td>
</tr>
<tr>
<td>Italic</td>
<td>Italic typeface indicates the title of a document, or signifies new terms.</td>
</tr>
<tr>
<td>Convention</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SMALL, BOLD CAPITAL LETTERS</td>
<td>Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, GET and CTRL.</td>
</tr>
<tr>
<td>KEY1+KEY2</td>
<td>A plus sign between key names indicates a <em>simultaneous</em> key sequence: you press and hold down the first key while pressing the second key. For example, CTRL+X.</td>
</tr>
<tr>
<td>KEY1 KEY2</td>
<td>A space between key names indicates a <em>sequential</em> key sequence: you press and release the first key, then press another key. For example, ESCAPE H.</td>
</tr>
</tbody>
</table>

**Syntax:**

| **Fixed width** | A fixed-width font is used in syntax, code examples, system output, and file names. |
| **Fixed-width italics** | Fixed-width italics indicate variables in syntax. |
| **Fixed-width bold** | Fixed-width bold italic indicates variables in syntax with special emphasis. |
| **UPPERCASE fixed width** | ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class. |
| **Period (.) or colon (:)** | All statements except DO, FOR, FUNCTION, PROCEDURE, and REPEAT end with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon. |
| **[ ]** | Large brackets indicate the items within them are optional. |
| [[]] | Small brackets are part of ABL. |
| {{}} | Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams. |
| {} | Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure. |
| | A vertical bar indicates a choice. |
| ... | Ellipses indicate repetition: you can choose one or more of the preceding items. |

**Example procedures**

OpenEdge documentation may provide example code that illustrates syntax and concepts. You can access many of the example files, and details for installing them, from the following locations:
A self-extracting Documentation and Samples file available on the OpenEdge download page of the Progress Software Download Center

- The OpenEdge Product Documentation Overview page on Progress Communities:


Once installed, you can locate the example files in the following paths under the OpenEdge Documentation and Samples installation directory:

<table>
<thead>
<tr>
<th>This directory . . .</th>
<th>Contains examples for the following documents . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>src\prodoc\dotnetobjects</td>
<td>OpenEdge Development: GUI for .NET Programming</td>
</tr>
<tr>
<td>src\prodoc\dynamics</td>
<td>The Progress Dynamics documentation</td>
</tr>
<tr>
<td>src\prodoc\getstartoop</td>
<td>OpenEdge Development: Object-oriented Programming</td>
</tr>
<tr>
<td>src\prodoc\handbook</td>
<td>OpenEdge Getting Started: ABL Essentials</td>
</tr>
<tr>
<td>src\prodoc\interfaces</td>
<td>OpenEdge Development: Programming Interfaces</td>
</tr>
<tr>
<td>src\prodoc\json</td>
<td>OpenEdge Development: Working with JSON</td>
</tr>
<tr>
<td>src\prodoc\langref</td>
<td>OpenEdge Development: ABL Reference</td>
</tr>
<tr>
<td>src\prodoc\prodatasets</td>
<td>OpenEdge Development: ProDataSets</td>
</tr>
<tr>
<td>src\prodoc\tranman</td>
<td>OpenEdge Development: Translation Manager</td>
</tr>
<tr>
<td>src\prodoc\visualdesigner</td>
<td>OpenEdge Getting Started: Introducing Progress Developer Studio for OpenEdge Visual Designer</td>
</tr>
<tr>
<td>src\prodoc\xml</td>
<td>OpenEdge Development: Working with XML</td>
</tr>
<tr>
<td>src\samples\open4gl\java</td>
<td>OpenEdge Development: Java Open Client</td>
</tr>
</tbody>
</table>

**OpenEdge messages**

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.

- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.

- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:
• Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.

• Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.

• Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.

• Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

If you encounter an error that terminates OpenEdge, note the message number before restarting.

**Obtaining more information about OpenEdge messages**

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

• Choose Help > Recent Messages to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.

• Choose Help > Messages and then type the message number to display a description of a specific OpenEdge message.

• In the Procedure Editor, press the HELP key or F1.

On UNIX platforms, use the OpenEdge pro command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

**To use the pro command to obtain a message description by message number:**

1. Start the Procedure Editor:

   ```
   OpenEdge-install-dir/bin/pro
   ```

2. Press F3 to access the menu bar, then choose Help > Messages.

3. Type the message number and press ENTER. Details about that message number appear.

4. Press F4 to close the message, press F3 to access the Procedure Editor menu, and choose File > Exit.
ADM Addenda

This chapter describes addenda to the ADM documentation since OpenEdge R10.1.

For details, see the following topics:

• Behavior change affecting procedure handle availability
• New DynCombo properties
• New DataView class
• Enabling columns for new records only
• Foreign values always applied when OpenOnInit is off
• Container publishes "createObjects" from createObjects
• Query manipulation on cached DataObjects
• Optimistic lock works for deletions
• NO-LOCK and read-only SDO tables
• EnableRule changes
• SDO always retrieves IndexInformation on first request
Behavior change affecting procedure handle availability

Introduced:
OpenEdge R10.1A
Due to changes to support SmartDataFields, the AllFieldHandles and AllFieldNames properties no longer store procedure handles and names for contained objects on static containers. This information is available in the ContainerTarget and InstanceNames properties.

In previous versions, handle lists were built in the container's initializeObject procedure. The handle lists are now built during the container's createObjects procedure.

New DynCombo properties

Introduced:
OpenEdge R10.1A
The DynCombo class has two new properties, AltValueOnAdd and AltValueOnRebuild. These properties control the displayed value in the combo when the appropriate value cannot be determined.

The AltValueOnAdd property controls what to display if the data source's initial value for the object that is adding a record does not exist in the combo list. The AltValueOnRebuild property controls what to display if the combo list is rebuilt during editing, for example, when the parent field's value is changed.

The following table lists the allowed values for both properties.

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Clear&gt;</td>
<td>Clears the combo and displays nothing. This is the default value.</td>
</tr>
<tr>
<td>&lt;First&gt;</td>
<td>Displays the first value in the list.</td>
</tr>
<tr>
<td>&lt;Last&gt;</td>
<td>Displays the last value in the list.</td>
</tr>
</tbody>
</table>

There is no tool support for this properties. You must set them directly in the code.

New DataView class

Introduced:
OpenEdge R10.1A
A data view is a logical view of data. The structure of a data view does not reflect the structure of data in a particular data source. Rather, the data view's structure represents a convenient arrangement of data for a particular task. A data view might include calculated data or even data from several different data sources.

The OpenEdge DataView is an ADM2 implementation of a data view. The DataView is based on ProDataSets to take advantage of their efficiency. To make integrating DataViews into your applications easier, DataViews have the same object type as SmartDataObjects (SDOs). This type enables existing visual objects to connect to DataViews exactly as they do to SDOs.

When you look at how a data object gets data from the data source to the UI, the process has these main parts:

- The interface between the data object and the UI
- The in-memory data storage
- The data access from the data source

In the SDO, these parts are encapsulated in a single object. When you work with DataViews, the process is spread between several objects. The DataView handles the data interface with the UI. The data interface transfers data between the UI and the in-memory data storage. One or more ProDataSets, managed by a DataContainer, handle the in-memory data storage. Finally, a Service Interface that you create serves as the gateway to the back-end process that handles data access with the ultimate data source.

This separation fits well with the OpenEdge Reference Architecture style of programming. The reference architecture also calls for a Service Interface between the back-end processes of the Business Services layer and the front-end processes of the Presentation and Integration layers. The DataView fits neatly into this prescription.

For more information on using DataViews, see the whitepapers in the Architecture, SaaS & Cloud Computing Community section on PSDN: http://communities.progress.com/pcom/community/psdn/openedge/architecture.

The ADM2 API is considered self-documenting. Please see the source code in the OpenEdge-install-dir\src\adm2 directory for details on new APIs and properties.

### Changes to the ADM2 hierarchy

The following figure shows the data object hierarchy in previous ADM versions.

**Figure 1: Hierarchy before DataView**
The following figure shows the new hierarchy.

**Figure 2: Hierarchy with DataView**

Changes in the query hierarchy moved many existing APIs. The first instance of some APIs moved up the hierarchy to a different file. In some of these cases, an override is left in the original file; in other cases, the API is removed from that file. Consult the source code if you need to find an API that moved.

In addition, some classes are subdivided into multiple superprocedures:

- **Query** — Contained in `query.p` and `queryext.p`
- **Data** — Contained in `data.p`, `datacols.p`, `dataext.p`, and `dataextapi.p`

### Query property changes

Some existing properties changed, as described in the following table.

**Table 2: New or changed properties**

<table>
<thead>
<tr>
<th>API</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoCommit</td>
<td>This property now has Read/Write capability.</td>
</tr>
<tr>
<td>KeyFields</td>
<td>This property is Read only in the DataView class.</td>
</tr>
</tbody>
</table>
When you specify sort direction in a BY phrase within this property, you can use a pseudo-keyword, TOGGLE. The setQuerySort function has internal logic to handle this word in a BY phrase. When the function finds TOGGLE in a BY phrase, the function checks the sort direction of the current BY phrase. The function applies the opposite sort direction to the new BY phrase.

**Caution:** TOGGLE is not a true Progress 4GL keyword. You cannot use this word in BY phrases outside the context of this function and a few related functions.

<table>
<thead>
<tr>
<th>API</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuerySort</td>
<td>When you specify sort direction in a BY phrase within this property, you can use a pseudo-keyword, TOGGLE. The setQuerySort function has internal logic to handle this word in a BY phrase. When the function finds TOGGLE in a BY phrase, the function checks the sort direction of the current BY phrase. The function applies the opposite sort direction to the new BY phrase.</td>
</tr>
<tr>
<td>UpdatableColumns</td>
<td>This property is Read/Write in the DataView class, but Read only in the Data class.</td>
</tr>
</tbody>
</table>

**Enabling columns for new records only**

**Introduced:**
OpenEdge R10.1C

SmartDataObjects and DataViews have a new UpdatableWhenNew property that accepts a comma-separated list of columns that can be updated only for new records.

DataVisual objects support a corresponding new EnabledWhenNew property that specifies visual fields that should be enabled only in add or copy mode. The DataVisual property inherits the value from the UpdateTarget by default, but can also be set to a comma-separated list of fields.

**Foreign values always applied when OpenOnInit is off**

**Introduced:**
OpenEdge R10.1C

Foreign values are now always applied on initialization. In previous versions, this did not happen when OpenOnInit was off. The ForeignValues would be added in dataAvailable when the parent navigated or as soon as any request was issued to the AppServer. But, a call to openQuery on a local connection could read all data. If you actually want to read all data in a child SDO, then the link to the data source should be removed.
Container publishes "createObjects" from createObjects

**Introduced:**

OpenEdge R10.1C

The ADM2 container now publishes "createObjects" from createObjects. Contained objects can subscribe to this event to ensure that their createObjects are done before initializeObject is called.

In Release 10.1C, any ADM2 object can now implement a createObjects method and add the method to its ContainerSourceEvents in order to create widgets or read Repository data before initializeObject is called.

The container class has always subscribed to this event and child containers in Progress Dynamics have always relied on this behavior to ensure that all objects are created before resizing takes place. However, this is a behavior change for ADM2 non-window containers, like SmartFrames and Viewers, since their static container typically did not publish the "createObjects" event beforehand and their createObjects were called from initializeObject.

If you have customized code that relies on a late call to createObjects, there are two options to fix the problem:

- Move the customization to initializeObject. This is the recommended solution. The assumption is that, if a customization fails when createObjects is called before initialization, then the customization really belongs in initializeObject.

- Unsubscribe these containers from the "createObjects" event. You can do this by adding the following code in the main block of the object. You could also add this code in containrcustom.i if it is general:

  ```plaintext
  RUN modifyListProperty IN TARGET-PROCEDURE
    (TARGET-PROCEDURE,
     'Remove':U,
     'ContainerSourceEvents',
     'createObjects').
  ```

  In environments with Progress Dynamics, this logic also needs checks to avoid that unsubscribing to the event which Progress Dynamics expects. For example, you could bypass the unsubscribe if getUseRepository is true.

Query manipulation on cached DataObjects

**Introduced:**

OpenEdge R10.1C

DataObjects that use caching now support client-side query manipulation. This enables Progress Dynamics filter support and enables you to use ADM2's filter source with cached SDOs.

Note that only criteria added with assignQuerySelection are applied to the cached SDO on the client. The addQueryWhere and setQueryWhere methods have no effect on the cached query.
**Optimistic lock works for deletions**

**Introduced:**
OpenEdge R10.1C

The optimistic conflict check is now done for deletions. Because of this, the record is refreshed from server and can be deleted on next attempt.

To accomplish the check, `fetchDBRowForUpdate` (which fetches the record and also performs the delete) now calls `compareDBrow`. In previous releases, `compareDbRow` was called after `fetchDBRowForUpdate` had completed. While these procedures generally are not intended for direct calls and local overrides, they can have general application-specific customizations which this change might affect. The most important change of behavior to consider is that `compareDBRow` now is called for deletes.

**NO-LOCK and read-only SDO tables**

**Introduced:**
OpenEdge R10.1C

You can use the `NoLockReadOnlyTables` property to specify a comma-separated list of read-only (non-updatable) tables. The specified tables remain NO-LOCKed during the transaction and are excluded from the optimistic lock check for changes of the SDO. Setting this property to 'ALL' specifies all the read-only tables in the SDO.

**EnableRule changes**

**Introduced:**
OpenEdge R10.1C

Several changes have been made to the toolbars' EnableRule.

**Find and filter actions EnableRule extended to disable when no parent**

The "Find" and "Filter2" actions used on the various Browse* toolbars, as well as the standard ADM2 "Filter" action, now check for "RecordState=RecordAvailable,NoRecordAvailable". The check ensures that the actions are disabled when no parent is available. The check identifies this by the `RecordState` in the target being "NoRecordAvailableExt".

For Progress Dynamics, "Find", "Filter", and "Filter2" are changed in the Repository. For standard ADM2, the `initAction` in `toolbar.p` is changed to add the rule to "Filter".
**New canUpdate() function for tableio and browsetoolbar actions**

All actions that can update data now include a `canUpdate()` function in their `EnableRule`. The `canUpdate()` function in the `dataview` class (adm2/datavis.p) reflects whether the object can update data. The change ensures that the "Delete" action is disabled when there is no `UpdateTarget`. This change was added to all actions for consistency.

For standard ADM2, the `initAction` in toolbar.p now has `canUpdate()` in the `EnableRule` for "Add", "Copy", "Delete", and "Update".

The following table lists the actions changed in the Progress Dynamic's Repository:

<table>
<thead>
<tr>
<th>Category</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tableio</td>
<td>&quot;Add&quot;, &quot;Copy&quot;, &quot;Delete&quot;, &quot;FolderUpdate&quot;, and &quot;Update&quot;</td>
</tr>
<tr>
<td>Other (for Browse* toolbars)</td>
<td>&quot;Add2&quot;, &quot;Copy2&quot;, &quot;Delete2&quot;, and &quot;Modify&quot;</td>
</tr>
</tbody>
</table>

Actions like "Save", "Reset", and "Undo" do not have the new function. Those actions already have rules to ensure that they are never enabled until an appropriate action has been executed.

**SDO always retrieves IndexInformation on first request**

*Introduced:*

OpenEdge R10.1C

In previous releases, the SDO retrieves the IndexInformation property from the server only when the SDO is batching. Now, the SDO always retrieves the property from the server on the first request. Also note that the property only applies to a single table. In previous releases, the property contained information on all tables joined in the SDO. You can override `getIndexInfoTables` if you want it to return information for more tables.

Because of the changes to the IndexInformation property, an SDO on a client without batching now retrieves more context information on the first request. The context information for even a single table might be large and affect performance.
Progress Dynamics Addenda

This chapter describes addenda to the Progress Dynamics® documentation since OpenEdge R10.1.

For details, see the following topics:

- Generated toolbars and translations
- New log file for Deploy Static Objects tool
- Decimal values in dynamic combo-boxes
- New 4GL Generator options
- Objects without deployment types
- Context help for SmartDataFields
- General Manager API behavior change
- Interrupting triggers on dynamic viewers
- Translating Filter/Find windows
- Toolbar and Menu Designer enhancement
- Relogon and Session Reset features deprecated
- Maximum tables in dynamic lookup query
- Widget IDs for ADM2 and Progress Dynamics
- Simplified message dialog for Progress Dynamics
- isObjQuoted function deprecated
• Global translations for TAB and TITLE elements
• Customizing AppBuilder traffic to the Repository
• Changes to instanceOf function
• Adding declarative statements in AppBuilder-generated code
• Multiple physical services for the same resource

Generated toolbars and translations

Introduced:
OpenEdge R10.1A

On the Options tab of the 4GL Generator tool, you can select whether or not to include translations in your generated objects. If you choose not to include translations in a generated toolbar, the toolbar incurs an extra AppServer hit to retrieve translations from the Repository. Typically, the toolbar will produce two AppServer hits, one for translation and one for security. This behavior might degrade performance compared to dynamically creating toolbars. The size and complexity of the toolbar affect whether a dynamic or static toolbar will yield the better performance.

New log file for Deploy Static Objects tool

Introduced:
OpenEdge R10.1A

The Deploy Static Objects tool now enables you to produce a listing file. When the tool packages a deployment, the file lists all the packaged files.

By default, the tool creates a file named listing.log in your working directory. If a filename is not specified, no listing file is created.

The file contains the following fields, tab-delimited:
• File name
• Relative path
• Deployment type
• Design-only flag

Decimal values in dynamic combo-boxes

Introduced:
OpenEdge R10.1A
In previous releases, you might encounter problems developing decimal dynamic combo-boxes. Previous releases stored the default flag values in the format used during development. This prevented deployment to numeric formats other than the one used for development. When deployed with another numeric format, either the combo displayed no data and generated errors or incorrect data was stored in the Repository for the <None> or <All> option.

Now default flag values are stored with American numeric format. You should enter the default flag values in the SmartDataField Maintenance Tool using the session’s format, but the tool replaces the numeric decimal point with a period in the stored value. At run time, any stored periods in the data are replaced with the current session's numeric decimal point.

The DCU upgrade runs a fix program to convert existing default flag values for decimal dynamic combos to American format. The DCU must run in the format that was used to develop dynamic combos.

The fix program writes a message to the log file for each default flag value it attempts to convert. It writes a message when a value is converted successfully or writes a message if the conversion fails. Review the DCU log file after the upgrade and manually correct any failures. Any failures that are not corrected might not behave properly at run time.

Note: There are several Progress Dynamics objects in the Repository that store blank default flag value instance attributes and give messages in the log file. These are for the cbSCMTool dynamic combo on the following viewer objects: gsmsxgenviewv, gsmsxotviewv, gsmsxpmviewv. The messages for these objects are expected and you can ignore them.

You must manually convert any dynamic combo instance attributes stored in static viewer code in your application to American numeric format.

New 4GL Generator options

Introduced:
OpenEdge R10.1A

The 4GL Generator tool enables you to create static equivalents of your dynamic objects to optimize performance. The 4GL Generator now enables you to set the GENERATE-MD5 and MIN-SIZE options for the r-code it generates.

Objects without deployment types

Introduced:
OpenEdge R10.1A

The Deploy Static Objects tool now enables you to specify what to do with objects that do not have a deployment type. The Include objects with blank deployment types toggle box controls whether or not these objects are included in your deployment. By default, this option is set and any object with a blank deployment type is included in the deployment. When the toggle is unchecked, these objects are excluded from the deployment.
Context help for SmartDataFields

Introduced:
OpenEdge R10.1A

When mapping context help in previous versions for SmartDataFields, there was a mismatch between the data required to create the mapping and the data required to retrieve the context help. In OpenEdge R10.1A, you map the help in the Map Context Help tool with the same data that you use to retrieve the help. The following table shows the new values for mapping SDF context help. These values are consistent with the values used to map context help for viewer datafields.

Table 3: Mapping SDF context help

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help container filename</td>
<td>Container name</td>
</tr>
<tr>
<td>Help object filename</td>
<td>Viewer name</td>
</tr>
<tr>
<td>Help fieldname</td>
<td>SDF FieldName</td>
</tr>
</tbody>
</table>

General Manager API behavior change

Introduced:
OpenEdge R10.1A

The updateTableViaSDO procedure in the General Manager has changed. In previous releases, this procedure ran SDOs directly. The procedure now starts SDOs using the startDataObject procedure. The procedure now can support updates with both static and dynamics SDOs. However, unlike in previous versions, the procedure cannot start an SDO that has not been registered in the Repository.

Interrupting triggers on dynamic viewers

Introduced:
OpenEdge R10.1A

You should use RETURN ERROR instead of RETURN NO-APPLY to interrupt triggers on dynamic viewers. RETURN NO-APPLY does not propagate up the procedure stack. This behavior means that it cannot interrupt a default event in a procedure that is defined as a trigger event for a widget on a dynamic viewer. An example of this situation is when you want to avoid setting focus on the next field on the TAB event.

In addition, RETURN ERROR works if the event procedure is defined directly as a persistent trigger outside of Progress Dynamics.
Translating Filter/Find windows

Introduced:
OpenEdge R10.1A

Providing translations for Progress Dynamics Filter/Find windows require several steps using the Translate window, the Menu Item Translation window, and the Message Control window.

A Filter/Find window contains the following translatable components:

- The window title
- Menu items and their tooltips
- Tab folder labels
- Radio-set labels
- Browse column labels

Note: The browse picks up any translations for database column labels that are added through the Entity Translation window.

- Labels of the widgets on the Advanced tab for filter windows
- The error message indicating when a permanent filter is set

Where to translate items

The following table lists the tools for translating items and which components you translate through those tools.

Table 4: Tools for translating window components

<table>
<thead>
<tr>
<th>Tool</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translate window</td>
<td>The window title</td>
</tr>
<tr>
<td></td>
<td>Tab folder labels</td>
</tr>
<tr>
<td></td>
<td>Radio-set labels</td>
</tr>
<tr>
<td></td>
<td>Browse column labels</td>
</tr>
<tr>
<td></td>
<td>Labels of widgets on the Advanced tab for filter windows</td>
</tr>
<tr>
<td>Menu Item Translation window</td>
<td>Menu items and their tooltips</td>
</tr>
<tr>
<td>Message Control window</td>
<td>Warning AF:149, the message when a permanent filter is set</td>
</tr>
</tbody>
</table>
Translating the window title

Providing a translation of the TITLE widget’s Original Label does not yield a properly translated window title. The window title consists of the actual name of the tool or table and one of two hidden fields, fiFilterLabel and fiFindLabel. The hidden fields provide the appropriate word, “Filter?” or “Find?”, depending on the mode in which the Filter/Find window is launched. That word is then added to the TITLE widget’s label to produce the final title.

To properly translate the window title, you must provide the following translations:

• A translation for fiFilterLabel
• A translation for fiFindLabel
• A translation for the TITLE widget

Note: You should only translate the words after the hyphen. The rest of the label is added as appropriate by the framework.

Toolbar and Menu Designer enhancement

Introduced:
OpenEdge R10.1A

You can now specify an internal procedure that is published just before the creation of a menu item or toolbar button. The Create event field in the Other section of an Item property sheet controls this behavior. The toolbar or menu looks for the procedure in the object specified in the Item link field. If no Item link is specified, the toolbar or menu looks in the container’s super procedure.

If you want to create a user-defined list of menu items, you can define the menu items using the defineAction function. For example, if you have a sub-menu called ‘Favorites’, you can call the defineAction function in the procedure specified in the Create event field to add child items, as shown in the following code:

DYNAMIC-FUNCTION('defineAction' in SOURCE-PROCEDURE,  
'Sol',  
'Sol'+CHR(1)+'Solitaire'+CHR(1)+'runSolitaire'+CHR(1)+'Favorites' +CHR(1) + 'RUN').

If you want to create a user-defined list of menu items that are re-constructed each time the menu is selected (for example, a list of opened windows), you can define a 'Menu drop function' for an item. This is a function located in the linked object defined by the ‘Item link’ (or container’s super if the item link is blank) that returns a CHR(1) delimited list of menu item names and captions.
Suppose you have a sub-menu called 'Modules' that changes with the context of the current record. You can add the following code to the function:

```plaintext
cTarget = DYNAMIC-FUNCTION('linkHandles':U IN TARGET-PROCEDURE, 'ContainerToolbar-Source':U) NO-ERROR.
hTarget = WIDGET-HANDLE(cTarget).
DYNAMIC-FUNCTION("defineAction" IN hToolbar, "Favorites", "OnChoose", "RunFavorites").
cList = "OE,Order Entry,IN,Inventory,CM,Customer Maintenance"
       = REPLACE(cList,"\",CHR(1)).
RETURN cList.
```

The procedure 'RunFavorites' defined on the parent item 'Favorites' would require a character input parameter. The system would pass the Key (odd entry in the list) to the procedure. (For more information, see the Working with the version 9 ADM: Using and Customizing the SmartToolbar whitepaper.)

---

**Relogon and Session Reset features deprecated**

**Introduced:**

OpenEdge R10.1C

The **Relogon** and **Session Reset** menu options are deprecated. They have been removed from the menus. However, the underlying APIs are still in the code for any user who has customized or extended these features.

---

**Maximum tables in dynamic lookup query**

**Introduced:**

OpenEdge R10.1C

In *OpenEdge Development: Progress Dynamics Basic Development*, the section on defining dynamic lookups states that a lookup's query can contain a maximum of 10 tables. The lookup's query can now contain a maximum of 18 tables.

---

**Widget IDs for ADM2 and Progress Dynamics**

**Introduced:**

OpenEdge R10.1C

When the OpenEdge GUI client renders widgets on a window, it calls a Microsoft Windows API during the widget creation. The Windows function allows the caller to pass a child-window identifier which Windows uses to identify that given widget. By default, the OpenEdge GUI client generates a new identifier for each widget each time the window runs. While this process is efficient, it prevents the use of third-party automated testing tools that rely on your widgets having the same unique identifier across sessions.
OpenEdge R10.1A introduced the **WIDGET-ID** attribute to enable you to assign a fixed identifier to the widgets on your interface. This attribute can provide third-party tools with the unique identifiers they require for automated testing. That release also included the **Widget ID Assignment Utility** PRO*Tool to enable you to assign widget IDs to simple GUI containers without opening them. However, that tool writes the widget ID directly into the static code for each widget which prevents it from assigning widget IDs to certain widgets.

**Note:** To make use of the **WIDGET-ID** attribute, you must start your session with the Use Widget ID (**-usewidgetid**) startup parameter. Using widget IDs causes extra processing to render your widgets. Since widget IDs are only intended for use with automated testing tools, you can avoid any performance impact by not specifying this parameter for your deployed applications.

OpenEdge R10.1C extends the support for widget IDs. New AppBuilder preferences enable automatic assignment of the **WIDGET-ID** attribute for certain widgets. This release also introduces the **Runtime WIDGET-ID Assignment Tool** PRO*Tool to enable you to easily assign widget IDs in ADM2 and Progress Dynamics applications. You can now assign widget IDs to the following:

- Widgets in dynamic objects, for example, a dynamic SmartDataViewer
- Dynamically-rendered widgets in SmartObjects, for example, a SmartSelect
- Multiple instances of a SmartObject in the same run-time container

**Note:** Some widgets are composed of elements that the third-party testing tools consider separate widgets. The widget IDs for these elements are always assigned by the AVM when the widget is rendered. For example, the AVM assigns widget IDs to the columns of a browse by incrementing the browse's widget ID by a constant for each column.

The **Runtime WIDGET-ID Assignment Tool** creates an XML file to store the widget IDs that it assigns. When the AVM renders the widgets at run time, it uses the numbers from this file to assign values to the widget's **WIDGET-ID** attributes. The XML file creates a context in which a dynamically-generated widget can have the same widget ID over several sessions.

As long as you do not change the number of widgets on a container, you can edit the container and get a repeatable set of unique widget IDs. However, if you add or remove a widget on a container, you must generate a new XML file. A given widget's widget ID might not match between the old and new files. Regenerating the file creates a new context, and you need to record your tests again in the third party tool to make use of these new widget IDs.

If you start a session without the Use Widget ID (**-usewidgetid**) startup parameter, the ADM rendering engine ignores the **WIDGET-ID** attribute and assigns identifiers through its default process. If you do specify the startup parameter, the ADM uses the following precedence to assign widget IDs:

1. The ADM reads the **widgetIDFileName** property for the container. If the ADM finds a filename, it uses the values in the XML file to assign the **WIDGET-ID** attribute for the listed widgets.
2. For the remaining widgets, the ADM determines if a **WIDGET-ID** attribute was specified in a static file. If the ADM finds a hard-coded value, it assigns that value to the widget.
3. For any remaining widgets, it assigns the next unused value, working from the top-end of the range backwards. The **WIDGET-ID** attribute accepts even INTEGER values of 2 through 65534.

**Note:** Widget IDs must be unique across a window. When a conflict occurs, a warning dialog appears and the ADM assigns the next available value as described in Step 3 above.
As this process shows, the actual value of a widget ID might be assigned by one of several methods. In general, the user does not need to know the actual value of a widget ID. The third-party tools access these numbers through the Windows API, rather than directly from OpenEdge. A user never needs to prepare a complete list of all the widget IDs in an application. The only time a user cares about the value is if a conflict arises because a newly assigned widget ID is not unique. The user can then alter the value of that widget to create a unique widget ID.

**AppBuilder preferences**

The AppBuilder **Preferences** dialog includes a **Widget ID** tab that enables you to control how the widget ID features operate:

![AppBuilder Preferences Dialog](image)

The following table describes this preference tab.

**Table 5: Widget ID Preferences**

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Automatically assign widget IDs</strong> toggle</td>
<td>When selected (the default), the AppBuilder automatically assigns <strong>WIDGET-ID</strong> attribute values to any static widget when it is created.</td>
</tr>
<tr>
<td><strong>Starting widget ID for frames</strong> fill-in</td>
<td>Defines the value for the first in a container when the AppBuilder assigns the <strong>WIDGET-ID</strong> attribute automatically.</td>
</tr>
<tr>
<td><strong>Frame widget ID increment</strong> fill-in</td>
<td>Defines the amount by which the AppBuilder increments the value for each additional frame in a container.</td>
</tr>
</tbody>
</table>
When selected (the default), the AppBuilder automatically assigns a value for the `widgetIDFileName` property when you first save a container. This setting has no effect after the first time you save the container. The AppBuilder stores the value in the static code or in the Progress Dynamics Repository, as appropriate.

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Save widget-id filename</strong> toggle</td>
<td>When selected (the default), the AppBuilder automatically assigns a value for the <code>widgetIDFileName</code> property when you first save a container. This setting has no effect after the first time you save the container. The AppBuilder stores the value in the static code or in the Progress Dynamics Repository, as appropriate.</td>
</tr>
<tr>
<td><strong>Default</strong> radio button</td>
<td>Defines the <code>widgetIDFileName</code> property as the default value, <code>container_filename.xml</code>. For example, the default XML file for the window, <code>wCust.w</code>, is <code>wCust.xml</code>.</td>
</tr>
<tr>
<td><strong>Custom XML file</strong> radio button</td>
<td>Displays a fill-in field where you can specify an alternate name for the XML file.</td>
</tr>
</tbody>
</table>

The preference options in the top box only apply when the AppBuilder automatically assigns widget IDs to the widgets in a static container. The preference options in the bottom box only apply to the XML files used by the Runtime WIDGET-ID Assignment Tool.

**Widget ID numbering**

The tools increment the widget IDs differently, depending on which tool you use. When assigning static widget IDs, the AppBuilder or **Widget ID Assignment Utility** use the following simple pattern:

1. The widget ID of the first frame in the container is the value specified in the **Starting widget ID for frames** preference.
2. The widget ID for the first widget in that frame is 2.
3. The widget ID for each additional widget in that frame increments by 2.
4. When all widgets in the frame have a widget ID, the tool moves to the next frame.
5. The tool sets the widget ID of the next frame by incrementing the widget ID of the previous frame by the value in the **Frame widget ID increment** preference.
6. The tool repeats Step 2 through Step 4 until all the widgets have widget IDs.
7. When the ADM renders the object, it adds the widget IDs of the widget and the frame together to produce the final widget ID value. For example, using the default values, the first widget in the first frame has a widget ID of 102; the second widget is 104; the first widget in the second frame is 202; the second widget is 204; and so on.

The **Runtime WIDGET-ID Assignment Tool** uses a different scheme when assigning widget IDs to widgets. Each object type has a different increment assigned in the `adecomm/_widgaps.p` procedure. As the tool assigns widget IDs in a container, it adds the widget ID of the preceding widget and the increment for the preceding widget's object type together to calculate the widget ID of the current object. Some widgets also act as containers for other widgets. For these widgets, the widget ID of the first contained widget is always zero plus the widget's increment. When the ADM renders the container at runtime, it assigns the final widget ID for the contained widgets by adding the container's widget ID to the widget ID of the contained widget. The following table lists the default increments used by the tool.
Table 6: Runtime WIDGET-ID Assignment Tool default increments

<table>
<thead>
<tr>
<th>Object type</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SmartContainer static FRAME</td>
<td>200</td>
</tr>
<tr>
<td>SmartBrowser or DynBrow</td>
<td>250</td>
</tr>
<tr>
<td>SmartViewer or DynView</td>
<td>1000</td>
</tr>
<tr>
<td>DataField (DynView only)</td>
<td>4</td>
</tr>
<tr>
<td>SmartDataField</td>
<td>20</td>
</tr>
<tr>
<td>DynCombo/DynLookup</td>
<td>10</td>
</tr>
<tr>
<td>SmartFilter</td>
<td>100</td>
</tr>
<tr>
<td>SmartFilter Field</td>
<td>6</td>
</tr>
<tr>
<td>SmartFrame</td>
<td>2000</td>
</tr>
<tr>
<td>SmartToolbar</td>
<td>200</td>
</tr>
<tr>
<td>SmartToolbarActions</td>
<td>2</td>
</tr>
<tr>
<td>SmartSelect</td>
<td>10</td>
</tr>
<tr>
<td>SmartFolder</td>
<td>1000</td>
</tr>
<tr>
<td>SmartFolder Page</td>
<td>20</td>
</tr>
<tr>
<td>SmartLOBField</td>
<td>10</td>
</tr>
<tr>
<td>SmartPanel</td>
<td>50</td>
</tr>
<tr>
<td>TreeNode</td>
<td>2000</td>
</tr>
</tbody>
</table>

**Note:** If these increments are not wide enough to create unique widget IDs for your application's widgets, you can customize the `adcomm/_widgaps.p` procedure as needed. However, remember that the WIDGET-ID attribute only accepts even INTEGER values of 2 through 65534.

For example, for a static SmartWindow with a SmartToolbar, SmartDataBrowser, SmartDataViewer, and a SmartFilter, the tool generates the following widget IDs:

1. The widget ID of the SmartWindow's default frame is 200, by default.
2. The tool adds the frame's widget ID, 200, to the frame's increment, 200, to create the next widget ID, 400. It assigns 400 as the widget ID of the first widget on the default frame, the SmartToolbar.
3. The tool adds the SmartToolbar's widget ID, 400, to its increment, 200, to create the next widget ID, 600. The SmartToolbar is a container for its SmartToolbarActions, which visualize as buttons. So, the buttons are on the SmartToolbar's frame, rather than the SmartWindow's default frame. The tool skips the buttons and assigns 600 to the next widget on the default frame, the SDB.
4. The SDB's widget ID and its increment, 250, makes the next widget ID, 850. The tool assigns 850 to the next widget on the default frame, the SDV.

5. The SDV's widget ID and its increment, 1000, makes the next widget ID, 1850. The tool assigns 1850 to the last widget on the default frame, the SmartFilter.

6. The tool moves to the next frame on the SmartWindow, the SmartToolbar's frame. The tool starts at 0 and adds 2, the increment for the first contained widget. Thus the first button's widget ID is 2 in the XML file. When the ADM renders the object, it adds the SmartToolbar's widget ID to the button's widget ID to get a final widget ID of 402.

**Note:** Menu options cannot be assigned widget IDs; only the buttons can have widget IDs.

7. The tool assigns widget IDs to all the buttons on the SmartToolbar and then moves on through the remaining frames. In this example, it assigns widget IDs for the columns in the SDB, the widgets on the SDV, and the fields on the SmartFilter.

### Using the Runtime WIDGET-ID Assignment Tool

You create widget ID files for your containers with the Runtime WIDGET-ID Assignment Tool.

To create a widget ID file:

1. Launch the container from another running container in the AppBuilder. This workaround bypasses a limitation caused by how the AppBuilder runs windows. For example, take this Customers window:
2. Click the Runtime WIDGET-ID Assignment button on the PRO*Tool palette. The Runtime WIDGET-ID Assignment Tool window appears:

4. Click **Add Containers to XML file**. The **Get Objects** dialog appears.

5. Choose **Runtime** and click **Get Objects**. The **Get Objects** dialog box displays all the running containers:

![Get Objects dialog box](image1)

6. Deselect the temporary AppBuilder version of the launch window and click **OK**. The tool assigns widget IDs to any widgets that do not have a widget ID and saves them to the XML file. It displays the widget IDs in a hierarchical browse:

![Runtime WIDGET T-ID Assignment Tool](image2)
7. If you change the container later, you can reopen the file and click **Synchronize XML data with source object** to update the widget IDs. The browse marks any objects that you added or removed from the container.

8. If you get a conflict warning for one of the widget IDs, you can enter a new value in the browse and save the new value back to the XML file.

---

**Simplified message dialog for Progress Dynamics**

**Introduced:**

OpenEdge R10.1C

OpenEdge R10.1C adds a method to control the visualization of the standard Progress Dynamics message dialog. A new Progress Dynamics session parameter, **MessageBoxType**, enables you to choose between three levels of visualization. So, without changing any objects in your application, you can change the functionality of the standard message dialog between different session types by changing the value of the session parameter.

By default, Progress Dynamics sessions continue to use the full functionality of the standard message dialog. None of the shipped session types include the **MessageBoxType** session parameter in their definitions. If you want to use simplified message dialogs, you should add the session parameter in your custom session types.

Users can override the session parameter in the user preferences. On the Administration window, choose **File > Preferences** to launch the **Dynamics Preferences** dialog. The **Message Box Type** combo-box enables users to choose the style of message box for their session.

The standard message dialog appears when the application code calls either the Session Manager's **showMessages( )** or **askQuestion( )** internal procedures. The Session Manager's **showWarningMessages( )** internal procedure launches a different dialog box. The new session parameter does not affect the visualization of this alternate message box.
Visualization levels

The MessageBoxType session parameter accepts the following values:

- **Simple** — Shows only the Message Summary of the standard message dialog:

  ![Simple Message Box Example](image1.png)

- **Detail** — Shows the Message Summary and adds a Details button. Clicking Details, expands the message box to show an editor containing the Message Detail information of the standard message dialog. For example:

  ![Detail Message Box Example](image2.png)

- **Complex** — Shows the standard message box. For example:

  ![Complex Message Box Example](image3.png)
Interaction with SESSION:DEBUG-ALERT

When SESSION:DEBUG-ALERT is TRUE, the standard message dialog includes a button to access an AVM stack trace through a secondary ABL message box. When you use either the Simple or Detail visualization level, the message dialog displays a similar button. While the icon is different, the functionality is the same.

isObjQuoted function deprecated

Introduced:
OpenEdge R10.1C

The isObjQuoted functions in the ADM2 smart.p class and the Progress Dynamics Session Manager have been deprecated. These functions were used internally by fixQueryString, which was deprecated in OpenEdge R10.1B.

Global translations for TAB and TITLE elements

Introduced:
OpenEdge R10.1C

Static containers can use global translations for TAB and TITLE elements, as well as named (container-specific) translations. The global translations are used on every window (for title) or every tab folder that has not been translated explicitly.

Customizing AppBuilder traffic to the Repository

Introduced:
OpenEdge R10.1C

Progress Dynamics code should not contain include references to adeuib, although calls to adeuib procedures can be made. However, in previous releases, some adeuib includes were erroneously added to the Progress Dynamics source code. OpenEdge R10.1C removes these include references.

The AppBuilder uses a procedure, ry/prc/rydynsckrp.p, to read the information that it needs to edit dynamic objects. It writes data back into the Repository using ry/prc/rygendynp.p. In previous releases, these procedures had references to include files in the adeuib directory. Because the adeuib source code is no longer shipped with the product, these references made it impossible to compile the procedures as released. The adeuib source code is still available from PSDN.

Release 10.1C moves the working code for reading and writing dynamic objects to adeuib/_dynsckr.p and adeuib/_gendyn.p, respectively. These procedures are called from rydynsckrp.p and rygendynp.p. All of the calls that the AppBuilder makes to this functionality are still through rydynsckrp.p and rygendynp.p (and not the adeuib procedures). This allows customers who have customized rydynsckrp.p or rygendynp.p to keep their customizations.
The only way to extend rydynsckrp.p or rygendynp.p is to modify the code itself; there is no defined way to extend these procedures. Note that customizing these procedures is not recommended. Customers who have existing customizations should overwrite rydynsckrp.p or rygendynp.p with their customized code. This code should be compared with the PSDN versions of _dynsckr.p or _gendyn.p to determine what has changed.

**Changes to instanceOf function**

**Introduced:**

OpenEdge R10.1C

The `instanceOf` function is extended to also work for non-Progress Dynamics applications. This enables class type checks to be done transparently across the two framework variants in the cases where the `ObjectType` does not provide sufficient separation. The `ObjectType` is still supported and used throughout the ADM2 and the AppBuilder.

The `instanceOf` in ADM2 uses the super procedure hierarchy. It considers the class name to be the super procedure name without the path and the `.p` extension. It is actually derived from the `&ADMsuper` preprocessor, which matches the super procedure name in all shipped super procedures. Progress Dynamics uses the actual class names in the Repository.

The `instanceOf` function also allows transparent references to classes that have different names in the two models, but really are the same class. For example, both `instanceOf('containr')` and `instanceOf('container')` can be used transparently in both environments. The following table lists names that can be used interchangeably:

<table>
<thead>
<tr>
<th>ADM2</th>
<th>Progress Dynamics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smart</td>
<td>Base (not likely to be used as all objects are of this type)</td>
</tr>
<tr>
<td>Combo</td>
<td>DynCombo</td>
</tr>
<tr>
<td>Lookup</td>
<td>DynLookup</td>
</tr>
<tr>
<td>Datavis</td>
<td>DataVisual</td>
</tr>
<tr>
<td>Containr</td>
<td>Container</td>
</tr>
</tbody>
</table>

Progress Dynamics has more classes than ADM2. References to classes that do not exist in ADM2, like "dynSDO", return false in ADM2.

The `instanceOf` function still checks the `ObjectType` as a last resort. This is not new behavior, but something that one should be aware of as there are classes in Progress Dynamics that have similar names to some of the `ObjectTypes`. When `instanceOf` checks with these names, it likely gives the desired effect because Progress Dynamics names are based on the ADM2, but you should use some caution. In particular, avoid using `instanceOf` to check for classes that begins with "Smart", as these names typically are classes in Progress Dynamics and `ObjectTypes` in ADM2.
Adding declarative statements in AppBuilder-generated code

Introduced:
OpenEdge R10.2B

Certain ABL statements (such as, USING and ROUTINE LEVEL ON ERROR UNDO, THROW) must come before any definitional and executable statement in order for a class or procedure to compile correctly. The AppBuilder's Procedure Settings dialog now enables you to specify statements of this type that you want generated into your AppBuilder code. The button launches the Edit Declarative Statements dialog with an editor where you can add these statements. The declarative statements added in the editor are generated into a new section in the source code. This section is written before both the optional temp-table definitions and the optional AppServer include.

Multiple physical services for the same resource

Introduced:
OpenEdge R10.2B

In some cases, you might need to set up multiple physical services in Progress Dynamics for the same resource. For example, if your application needs both synchronous and asynchronous connections to the same AppServer, you might set up Progress Dynamics physical services for each type of connection, using different connection parameters to ensure that there are two different handles. You might also consider creating different AppServices on the AppServer instance for the two connection types.