Getting started with DateTime

November 2005

LANSA®
ADVANCED SOFTWARE MADE SIMPLE

The information contained in this document represents the current view of LANSA on the issues discussed as of the date of publication. Because LANSA must respond to changing market conditions, it should not be interpreted to be a commitment on the part of LANSA, and LANSA cannot guarantee the accuracy of any information presented after the date of publication. In many cases, information in this document is dependent on information from third-party vendors.

The statements in this document about specific product features represent the current status or planned intentions of LANSA within one year of publication.

LANSA development plans are subject to change or withdrawal without further notice. Any reliance on this document is at the relying party's sole risk and will not create any liability or obligation for LANSA.

This document is for informational purposes only. **LANSA MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, IN THIS DOCUMENT.**

V1.1

The Americas:
Headquarters – Chicago, USA
Tel: +1 630 874 7000
Email: info@lansa.com

Europe:
Headquarters – London, UK
Tel: +44 1727 790300
Email: info@lansa.co.uk

Asia Pacific:
Headquarters – Sydney, Australia
Tel: +61 2 8907 0200
Email: info@lansa.com.au

www.lansa.com

Table of Contents

Getting started with DateTimes

This document is intended to give you some practical advice about the use of DateTime fields. It talks about their mechanics, the do's and don'ts of incorporating them into your database. It also talks about how to write RDMLX code to manipulate them in your components, WAMs and functions. It provides examples of RDMLX code. It's just about everything you need to know to get started.

Don't Panic

DateTime stores a Date and Time in the one field, to represent a point in time. You could avoid using DateTime and just use a Date field and a Time field, as together, these can do the same thing. But a DateTime field has one distinct difference: LANSA manages DateTime fields so that if you move the data from one time zone to another, the DateTime always allows data to be represented in the local time zone or in UTC. By default, LANSA displays DateTimes in the local time zone, but saves them in the database in UTC.

This may all sound rather technical, mysterious and difficult. What it actually means is that you can write applications that handle a point in time neatly, no matter what time zones your users are in when they enter or retrieve data. Once you're aware of a few do's and don'ts, using them will become second nature, just like using any other field type.

What is UTC?

UTC (Coordinated Universal Time) is the modern name for GMT (Greenwich Mean Time). It takes into account the addition or omission of leap seconds by atomic clocks each year to compensate for changes in the rotation of the earth.

Where Else to Look

The Field Types section of the Technical Reference Guide will provide you with some rules about how DateTimes can be used. You should read it once you've finished reading this document.

Do you care about Time zones?

If all of your application will always be executing in a single time zone, you really don't need a DateTime. You can just use a Date and Time, and avoid reading the rest of this document. If you really want to use a DateTime, then read on.

I really want to use a DateTime

Need to know:

- LANSA expects a DateTime field to be a UTC time. You can put any time zone DateTime into the field, but if the time zone is not UTC, you must be very careful how the DateTime field is handled.
- The SUTC (store in UTC) field attribute is on by default. The DUTC (display as UTC) field attribute is off by default. This means that if you put the DateTime on a file, the value in database (UTC) will not match value on screen (local time), unless your time zone is the same as UTC.
- The default attributes for new DateTime fields can be modified in the partition definition. After reading this document, you may decide not to use the LANSA-shipped default attributes for DateTime. You should modify the partition defaults for DateTime to match your DateTime handling strategy.
- DateTime Literals are always in UTC if they are in the YYYY-MM-DD HH:MM:SS[.fffffffff] format. (Refer to the Technical Reference guide for other literal formats.)
- Use *SQLNULL or *NULL as the field default, rather than a system variable such as *DATETIME.
- Read the following sections for more specific information.

My application will always run in a single time zone

First, are you sure? Here are some points to double-check:

- If your application is on the Internet, it can be executed from anywhere in the world, and therefore in different time zones.
- If your application interacts with LANSA Integrator, the service you are interacting with may be in a different time zone.
- If your application is a package that may be sold to many customers, one of those customers may need to run in multiple time zones.

The following is only relevant if your application is running in a single time zone.

If you are confused by the DateTime values in the database being different to the values on the screen, you have two alternatives. First, you can enable the DUTC flag:

- The intention is to ignore the fact that time zones exist and treat UTC and the local time zone as the same.
- Fields contain local time values, not UTC. The debugger shows these local time values.
- Data entered on the screen will be identical to that saved in the database.
- Literals match the values saved in the database.
- *DATETIME and *DATETIMEC can be used to assign the current time to the DateTime field. The .Now() intrinsic function must NOT be used.
- This is NOT recommended by LANSA, as if your application later needs to support multiple time zones, you will need to disable the DUTC and SUTC attributes on your fields to allow for the fact that the DateTime data in your files is actually stored and displayed in local time, not UTC. You will also need to rebuild and retest all parts of your application that work on DateTime fields.

The second alternative to reduce confusion is to disable the SUTC flag.

- Fields, correctly, contain UTC values. The debugger shows these UTC values.
- Data entered on the screen will be identical to that saved in the database.
- Literals do not match the values saved in the database.
- *DATETIME and *DATETIMEC must NOT be used, as they are in local time. A future Version 11.0 EPC will support setting the *DATETIME_UTC system variable to assign the current UTC time to a DateTime field. Until then, use the .Now() intrinsic function. For example: #MYDTFLD.Now().
- This alternative is fully supported by LANSA in a single time zone, as the lack of SUTC and DUTC attributes correctly indicates that the DateTime data in your files is actually stored and displayed in local time. However, disabling SUTC is still not recommended, as it is difficult to predict whether your application may change in the future and need to run in multiple time zones. The only exception is if all database updates are done by the server, for example using LANSA SuperServer to update the database.

My application needs to run in multiple time zones

- Leave the SUTC attribute enabled. If you want your users to maintain the data in UTC, then enable the DUTC attribute. For example, this might be appropriate for a LANSA function that will be executing in a Web browser.
- Fields, correctly, contain UTC values. The debugger shows these UTC values.
- Literals match the values saved in the database.
- *DATETIME and *DATETIMEC must NOT be used, as they are in local time. In a future Version 11.0 EPC, the system variable *DATETIME_UTC will be available to assign the current time in UTC to a DateTime field. In the meantime, use the .Now() intrinsic function. For example: #MYDTFLD.Now().
- Create a sample file with a DateTime field on it and write some application code to manipulate the field, to ensure you understand its behavior.
- This is LANSA's recommended alternative for DateTime handling.

Virtual DateTime Fields

Can be initialized with a code fragment of .Now(), and used to initialize real fields.

Other Files

When a DateTime field is created by loading an Other File, it is assumed that the Other File stores data in local time, and that you want to display it in local time. Therefore, neither the SUTC or DUTC attribute is enabled on the field. You must adjust the attributes if the assumption is incorrect.

SELECT_SQL: Using DateTime in WHERE clause

If you wish to create a WHERE clause which will use a condition on a DateTime column on your table, you need to follow two rules:

1. When comparing a DateTime column to another field, the field used must be a reference field of the column. For example:

```
DEFINE #DTIMX REFFLD (#STD_DTIMX) DEFAULT(*NULL)
…
SELECT_SQL … WHERE ('STD_DTIMX > :DTIMX')
```

2. When comparing a DateTime column to a literal, the literal must be formatted appropriately for the database that will be processing it, and according to the SUTC attribute on the field. It is much easier to use a field for comparison purposes as in rule 1, as you will not need to worry about the time zone conversion or the formatting of the DateTime string, as LANSA does it all for you.

   For example, to find all rows where #STD_DTIMX is greater than the local time value entered by a user in Sydney of 2005-09-06 12:05 PM (time zone is +10 hours from UTC), we need to get the value into a string in UTC, and then reformat appropriately for each platform (code has been simplified):

```
CASE_OF FIELD(#CPUTYPE)
WHEN  VALUE_IS('= WINNT')
   #STD_TEXTL := 'STD_DTIMX > { ts ''2005-05-06 02:05:00'' }'
WHEN  VALUE_IS('= AS400')
   #STD_TEXTL := 'STD_DTIMX > ''2005-05-06-02.05.00'' '
WHEN  VALUE_IS('= UNIX')
   #STD_TEXTL := 'STD_DTIMX > ''2005-05-06 02:05:00'' '
OTHERWISE
   ABORT …
ENDCASE
SELECT_SQL … WHERE(#STD_TEXTL)
```