Gute Frage 👍 – warum überhaupt der Aufwand mit „Ladeort"/„Ladeadresse", wenn man ja schon SHP (Absender) hat?

Hintergrund / Nutzen

- **Absender ≠ Ladeort**
  Sehr oft stimmt die Absenderadresse nicht mit dem tatsächlichen Ladeplatz überein.
  - Beispiel: Absender ist die **Zentrale in München**, geladen wird aber **im Außenlager in Augsburg**.
  - Ohne Ladeort fährt der Fahrer also zur falschen Adresse.
- **Praxis der Speditionen**
  Viele Transportaufträge laufen so:
  - Auftraggeber/Absender: Firma A (Headquarter)
  - Ladeort: Werk B, Lagerhalle C, Rampe D
  - Entladeort: wieder eine andere Adresse
- **EDI / Fortras / EDIFACT**
  Deshalb gibt es in fast allen Standards die Möglichkeit, **Shipper (SHP)** und **Loading Place (LDP/LOC)** getrennt zu codieren.
  → So weiß der Empfänger des Datensatzes, **wo er wirklich hinfahren muss**.

Vorteile in der IT-Umsetzung

1. **Weniger Missverständnisse**: Absender in München, Ladeort in Augsburg → klar getrennt.
2. **Automatische Navigation**: Ladeadresse kann direkt ans Navi oder an eine App übergeben werden.
3. **Zeitslots / Rampenplanung**: Zeitfenster hängen am Ladeort, nicht am Absender.
4. **Mehrere Ladeorte pro Auftrag** möglich (z. B. Sammelaufträge).

Fazit

Man braucht das immer dann, wenn der **tatsächliche Ladeplatz** von der **Adresse des Auftraggebers/Absenders** abweicht.
Im Alltag der Spedition ist das eher die **Regel als die Ausnahme** – darum ist die Ladeadresse im Fortras-/EDI-Umfeld quasi Pflichtfeld, sobald es komplexer wird.

Perfekt – dann ein realistisches **Speditionsbeispiel**:

Szenario

- **Auftraggeber / Absender (SHP)**:
  Muster GmbH, Zentrale München
- **Ladeort (ADR/LOC LDP)**:
  Werk Augsburg, Rampe 3
- **Entladeort / Empfänger (CNE)**:
  Beispiel AG, Lager Wien

Fortras-ähnlicher Aufbau (vereinfacht)

```
B00|REF=4711|DAT=20250929
SHP|NAM=Muster GmbH|STR=Zentrale 1|ZIP=80331|CTY=München|CTR=DE
ADR|TYP=LDP|NAM=Werk Augsburg Rampe 3|STR=Industriestr 12|ZIP=86150|CTY=Augsburg|CTR=DE
LOC|TYP=LDP|CODE=DEAGB   // UN/LOCODE optional
CNE|NAM=Beispiel AG|STR=Lagerstr 7|ZIP=1020|CTY=Wien|CTR=AT
```

Erklärung

- **B00** → Kopfdaten: Referenz + Datum
- **SHP** → Absender (München, Zentrale)
- **ADR mit TYP=LDP** → Ladeort (Augsburg, Rampe 3)
- **LOC mit TYP=LDP** → standardisierter Ortscode (optional, UN/LOCODE)
- **CNE** → Empfänger (Wien)

Warum wichtig

- Ohne den **ADR LDP** würde der Fahrer nach München (Zentrale) geschickt – obwohl dort gar keine Rampe ist.
- Mit Ladeort ist eindeutig klar: **geladen wird in Augsburg**, unabhängig davon, wer der Auftraggeber ist.

Das klingt nach einer sehr spezifischen Frage zur **Fortras-Schnittstelle** (FORmular TRAnsport System) im Speditions-/Logistikbereich. Ich versuche, dir die Richtung zu geben:

In **Fortras 100/200/300** gibt es keine eigene Feldgruppe nur für „Ladeadresse“, sondern die Ladeadresse wird normalerweise in den **Shipper-/Consignee-Blöcken (SHP, CNE)** oder über zusätzliche Ortsangaben abgebildet.

👉 Typische Möglichkeiten:

- **B00 (Allgemeine Kopfdaten)** enthält die Bezugsschlüssel, aber nicht die Ladeadresse selbst.
- **SHP (Shipper/Absender)** – hier ist oft der „Ladeort“ identisch mit dem Absender.
- Falls Ladeort ≠ Absender: dann kommt ein eigener Block ins Spiel, z. B.
  - **LDP (Ladeplatz / Place of Loading)** → in manchen Fortras-Erweiterungen als Zusatzblock vorgesehen.
  - Alternativ: **ADR-Zusatzfelder** im SHP-Segment.

💡 Viele Implementierungen lösen das so:

- **SHP** = Auftraggeber/Absender.
- **Ladeort** wird als eigener **ADR-Eintrag** mit Kennung „LDP“ oder per **Ort-Code** (UN/LOCODE) im SHP-Block angehängt.
- In neueren Fortras-Versionen (300, EDIFACT-ähnlich) gibt es **LOC-Segmente**, die explizit Lade-/Entladeorte kodieren.

📌 Fazit für deine Frage:
Du siehst den Wald nicht vor lauter Bäumen 😊 – die Ladeadresse ist **kein eigener Grundblock**, sondern hängt in der Praxis an **SHP** (Absender) oder wird über ein Zusatzsegment **LDP/LOC** angegeben.
Wenn der Ladeort vom Absender abweicht → eigenen Adressblock verwenden (in manchen Code-Tabellen als **ADR mit Code „LDP“** vorgesehen).

Kurz: **Nein, das ist nicht kompliziert.**
Wenn du SHP (Absender) schon hast, ist „Ladeort“ meist nur **1 zusätzlicher Adressblock + 1–2 Mapping-Regeln**. Aufwand: überschaubar (Formfeld, Validierung, Export/Import-Mapping).

Praktischer Fahrplan (bewährt)

1. **DB/Model**: optionales `loading_address` (Name, Straße, PLZ, Ort, Land, Kontakt, UN/LOCODE).
2. **UI**: Checkbox „Ladeort ≠ Absender“ → dann Felder einblenden.
3. **Regel**: Wenn leer ⇒ beim Export **SHP** auch als Ladeort interpretieren. Wenn gefüllt ⇒ **eigener Block**.
4. **Export-Mapping**:
   - Variante A (einfach): in **SHP** belassen, aber `OrtTyp=Ladeort` kennzeichnen.
   - Variante B (sauber): **ADR mit Code LDP** (Loading Place) bzw. **LOC(Ladeort)** je nach Partnerprofil.
5. **Import**: Falls ADR/LOC mit LDP kommt ⇒ in `loading_address` mappen, sonst auf SHP rückfallen.

Minimal-Beispiel (pseudomäßig, gut für Fortras-ähnliche Flatfiles)

```
B00|REF=4711|DAT=20250929
SHP|NAM=Beispiel GmbH|STR=Hauptstr 1|ZIP=80331|CTY=München|CTR=DE
ADR|TYP=LDP|NAM=Ramp A Lager West|STR=Industriestr 5|ZIP=80939|CTY=München|CTR=DE
LOC|TYP=LDP|CODE=DEMUC
```

- **ADR TYP=LDP** = Ladeadresse
- **LOC TYP=LDP** (optional) = Ladeort-Code (z. B. UN/LOCODE)

Mini-Export (PHP-artig)

```
function exportFortras(array $s) {
  $out = [];
  $out[] = "B00|REF={$s['ref']}|DAT={$s['date']}";
  $out[] = "SHP|NAM={$s['shipper']['name']}|STR={$s['shipper']['street']}|ZIP={$s['shipper']['zip']}|CTY={$s['shipper']['city']}|CTR={$s['shipper']['c
  if (!empty($s['loading'])) {
    $L = $s['loading'];
    $out[] = "ADR|TYP=LDP|NAM={$L['name']}|STR={$L['street']}|ZIP={$L['zip']}|CTY={$L['city']}|CTR={$L['country']}";
    if (!empty($L['unlocode'])) $out[] = "LOC|TYP=LDP|CODE={$L['unlocode']}";
  }
  return implode("\n", $out)."\n";
}
```

Mini-Import (erkenne Ladeort)

```
function parseFortras(array $lines) {
  $res = ['shipper'=>[], 'loading'=>null];
  foreach ($lines as $ln) {
    $p = explode('|', trim($ln));
    $seg = $p[0] ?? '';
    $kv = [];
    foreach (array_slice($p,1) as $pair) { [$k,$v] = array_pad(explode('=',$pair,2),2,''); $kv[$k]=$v; }
```

```
    if ($seg==='SHP') $res['shipper'] = $kv;
    if ($seg==='ADR' && ($kv['TYP']??'')==='LDP') $res['loading'] = [
      'name'=>$kv['NAM']??'', 'street'=>$kv['STR']??'', 'zip'=>$kv['ZIP']??'',
      'city'=>$kv['CTY']??'', 'country'=>$kv['CTR']??''
    ];
    if ($seg==='LOC' && ($kv['TYP']??'')==='LDP' && $res['loading']) $res['loading']['unlocode']=$kv['CODE']??'';
  }
  return $res;
}
```

Edge Cases & Tipps

- **Zeitfenster** (From/To): optional als `DTM|TYP=LDP|FROM=20250929T0800|TO=20250929T1200`.
- **Partnerprofile**: Manche wollen statt `ADR/LOC` alles in **SHP** – löse das mit einer **Mapping-Tabelle pro Partner**.
- **Validierung**: Wenn `loading` existiert → `name, zip, city, country` Pflicht, optional `unlocode`.

Hier ist ein kompaktes **Harbour / xBase++**-Beispiel, das einen Fortras-ähnlichen Datensatz mit **SHP (Absender)**, **ADR TYP=LDP (Ladeort)**, optional **LOC TYP=LDP**, und **CNE (Empfänger)** erzeugt und wieder einliest.

> Hinweis: Ich nutze `hb_ATokens()` (Harbour). In xBase++ kannst du dasselbe Prinzip mit `Token()`/`StrToken()` (oder eigener Split-Funktion) umsetzen. Der Rest ist 1:1 übertragbar (klassische Prozeduren/Funktionen, keine Harbour-Spezialobjekte nötig).

```
/* demo_fortras.prg
   Build + Parse eines Fortras-ähnlichen Flatfiles (B00/SHP/ADR:LDP/LOC:LDP/CNE)
   Getestet mit Harbour; für xBase++: ersetze hb_ATokens() durch deine Split-Funktion.
*/

PROCEDURE Main()
   LOCAL cOutFile := "fortras_demo.txt"
   LOCAL cTxt, hParsed

   // --- Beispiel-Daten (realistisches Szenario) ---
   LOCAL hB00 := {=>}
   hB00["REF"] := "4711"
   hB00["DAT"] := "20250929"

   LOCAL hSHP := {=>}
   hSHP["NAM"] := "Muster GmbH"
   hSHP["STR"] := "Zentrale 1"
   hSHP["ZIP"] := "80331"
   hSHP["CTY"] := "München"
   hSHP["CTR"] := "DE"

   LOCAL hLDP := {=>}     // Ladeadresse (ADR TYP=LDP)
   hLDP["NAM"] := "Werk Augsburg Rampe 3"
   hLDP["STR"] := "Industriestr 12"
   hLDP["ZIP"] := "86150"
   hLDP["CTY"] := "Augsburg"
   hLDP["CTR"] := "DE"

   LOCAL hLOC := {=>}     // optional: standardisierter Ort (UN/LOCODE)
   hLOC["CODE"] := "DEAGB"

   LOCAL hCNE := {=>}     // Empfänger
   hCNE["NAM"] := "Beispiel AG"
   hCNE["STR"] := "Lagerstr 7"
   hCNE["ZIP"] := "1020"
   hCNE["CTY"] := "Wien"
   hCNE["CTR"] := "AT"

   // --- Export: baue Text ---
   cTxt := ""
   cTxt += BuildSeg( "B00", hB00 )
   cTxt += BuildSeg( "SHP", hSHP )
   cTxt += BuildADR_LDP( hLDP )             // ADR|TYP=LDP|...
   cTxt += BuildLOC_LDP( hLOC )             // LOC|TYP=LDP|...
   cTxt += BuildSeg( "CNE", hCNE )

   MemoWrit( cOutFile, cTxt )
   ? "Datei erzeugt:", cOutFile
   ? "------"
   ?? cTxt

   // --- Import: parse zurück in Struktur ---
   hParsed := ParseFortras( cTxt )
   ? "------"
   ? "Parsed summary:"
   ? "REF:",      hb_HGetDef( hb_HGetDef( hParsed, "B00", {=>} ), "REF", "" )
   ? "SHP.City:", hb_HGetDef( hb_HGetDef( hParsed, "SHP", {=>} ), "CTY", "" )
   ? "LDP.Zip:",  hb_HGetDef( hb_HGetDef( hParsed, "LDP", {=>} ), "ZIP", "" )
   ? "LOC.Code:", hb_HGetDef( hb_HGetDef( hParsed, "LOC", {=>} ), "CODE", "" )
   ? "CNE.Name:", hb_HGetDef( hb_HGetDef( hParsed, "CNE", {=>} ), "NAM", "" )

   RETURN
//-------------------------------------------------------------

/* BuildSeg()
   Allgemeiner Segment-Builder: TAG|K1=V1|K2=V2 ...
*/
FUNCTION BuildSeg( cTag, hKV )
   LOCAL c := cTag, aKeys, i, cKey, cVal

   aKeys := hb_HKeys( hKV )
   FOR i := 1 TO Len( aKeys )
      cKey := aKeys[i]
      cVal := Any2Str( hb_HGet( hKV, cKey ) )
      c   += "|" + cKey + "=" + EscapeVal( cVal )
   NEXT
```

```
   RETURN c + hb_eol()

/* BuildADR_LDP()
   Spezieller Komfort-Builder für ADR mit TYP=LDP (Ladeadresse)
*/
FUNCTION BuildADR_LDP( hAdr )
   LOCAL h := hb_HClone( hAdr )
   hb_HSet( h, "TYP", "LDP" )
   RETURN BuildSeg( "ADR", h )

/* BuildLOC_LDP()
   Optionaler LOC-Eintrag (z.B. UN/LOCODE) für den Ladeort
*/
FUNCTION BuildLOC_LDP( hLoc )
   IF Empty( hLoc ) .OR. Empty( hb_HGetDef( hLoc, "CODE", "" ) )
      RETURN ""   // nichts ausgeben, wenn kein Code
   ENDIF
   LOCAL h := hb_HClone( hLoc )
   hb_HSet( h, "TYP", "LDP" )
   RETURN BuildSeg( "LOC", h )

/* ParseFortras()
   Liest den Text und liefert eine Hash-Struktur mit Schlüsseln:
   "B00", "SHP", "LDP" (aus ADR TYP=LDP), "LOC" (TYP=LDP), "CNE"
*/
FUNCTION ParseFortras( cTxt )
   LOCAL hRes := {=>}, aLines, i, cLine, cTag, aParts, j, aKV, cK, cV, hTmp, cTyp

   aLines := hb_ATokens( cTxt, hb_eol() )
   FOR i := 1 TO Len( aLines )
      cLine := AllTrim( aLines[i] )
      IF Empty( cLine )
         LOOP
      ENDIF

      aParts := hb_ATokens( cLine, "|" )
      cTag   := Upper( aParts[1] )

      // key=value Paare einsammeln
      hTmp := {=>}
      FOR j := 2 TO Len( aParts )
         aKV := hb_ATokens( aParts[j], "=" )
         cK  := IIF( Len(aKV)>=1, Upper(AllTrim(aKV[1])), "" )
         cV  := IIF( Len(aKV)>=2, UnescapeVal( aKV[2] ), "" )
         IF !Empty( cK )
            hb_HSet( hTmp, cK, cV )
         ENDIF
      NEXT

      DO CASE
      CASE cTag == "B00"
         hb_HSet( hRes, "B00", hTmp )
      CASE cTag == "SHP"
         hb_HSet( hRes, "SHP", hTmp )
      CASE cTag == "ADR"
         cTyp := Upper( hb_HGetDef( hTmp, "TYP", "" ) )
         IF cTyp == "LDP"
            hb_HSet( hRes, "LDP", hTmp )   // Ladeadresse
         ENDIF
      CASE cTag == "LOC"
         cTyp := Upper( hb_HGetDef( hTmp, "TYP", "" ) )
         IF cTyp == "LDP"
            hb_HSet( hRes, "LOC", hTmp )   // Ladeort Code
         ENDIF
      CASE cTag == "CNE"
         hb_HSet( hRes, "CNE", hTmp )
      OTHERWISE
         // Unbekannte Segmente bei Bedarf ablegen:
         // hb_HSet( hRes, "EXT_"+cTag, hTmp )
      ENDCASE
   NEXT

   RETURN hRes

/* --- Utilitys --- */

STATIC FUNCTION Any2Str( u )
   LOCAL c := ""
   DO CASE
   CASE ValType( u ) == "C" ; c := u
   CASE ValType( u ) == "N" ; c := LTrim( Str( u ) )
   CASE ValType( u ) == "D" ; c := DToS( u )
   CASE ValType( u ) == "L" ; c := IIF( u, "1", "0" )
   OTHERWISE                ; c := AllTrim( hb_ValToExp( u ) )
   ENDCASE
   RETURN c

/* EscapeVal / UnescapeVal
   Einfache Maskierung für "=", "|" und Steuerzeichen, falls nötig.
   (Hier minimal: ersetze Pipe und CR/LF)
*/
STATIC FUNCTION EscapeVal( c )
   c := StrTran( c, "|",  "\|" )
   c := StrTran( c, Chr(13), "\r" )
   c := StrTran( c, Chr(10), "\n" )
   RETURN c

STATIC FUNCTION UnescapeVal( c )
   c := StrTran( c, "\n", Chr(10) )
   c := StrTran( c, "\r", Chr(13) )
   c := StrTran( c, "\|", "|" )
   RETURN c
```

Was du anpassen kannst

- **xBase++ Port**: Tausche `hb_ATokens()` gegen eine Tokenizer-Funktion deiner Wahl (z. B. `StrToken()` in einer Schleife). Die restliche Logik (Stringaufbau, `MemoWrit`) bleibt identisch.
- **Partnerprofile**: Wenn ein Partner **kein ADR/LOC** will, kannst du `BuildADR_LDP()`/`BuildLOC_LDP()` weglassen und alles in `SHP` mappen. Die Struktur bleibt flexibel.
- **Zeitfenster**: Ergänze z. B. `DTM|TYP=LDP|FROM=20250929T0800|TO=20250929T1200` analog zu `BuildSeg()`.

Wenn du magst, baue ich dir noch eine **kleine Validierung** (Pflichtfelder für LDP: `NAM, ZIP, CTY, CTR`) oder ein **Partner-Mapping** (Schalter: „LDP als eigenes ADR vs. in SHP integrieren“).