XTend generator

Document: XTend generator – Concept document

Author: Holger Scherer, RZKH GmbH – hs@rzkh.de / www.rzkh.de

Version: v0.02

Date: 2020-10-27

Status: work in progress

Abstract: This document is a first draft of the conceptional idea on „XTend“ – a tool for creating business apps based on a definition, using a separate creator routine, without a specific platform or language in mind.

Table of contents

table of contents

Introduction to the project

What is „XTend“?

In first sight, XTend is just an idea on how to create business applications without a specific platform or programming language in mind. Based on working and creating business application for customers for a lot of years, an idea came into my mind some ten years ago: most business applications have common ideas and principles:

- there are two main types of screens: list (master) and work (detail) screens
- there is a menu based on access rights (groups or individuals)
- the look and feel usually is the same for the main screen types
- most screens have some logic to check and validate user entry or calculate data
- most programmers use templates for creating their screen programs which may be very old. If one wants to change the general behavior of all programs, there is a lot of work to do – and modifying 100 programs result in 100 potential errors – at least.
- for sure there is some exceptions: calling batch programs for data processing or complex data verification, printing, communications etc.

So programmers started to make routines for creating the same type of screen again and again based on a template (which might be updated over the time). Or the programmers started to move display logic to a source separated from the business logic, which resulted in a lot of different architectures and the usage of different languages which rose up in the market.

Programmers were forced to rewrite parts of the whole of their applications, as platforms or languages changed or were considered “outdated”. So they reinvented the same business logic using a new language, or used this situation to rewrite their applications with new logic, or just bought a new software (the easy way?).

The idea with XTend is to move most of the UI design and all (easy) business logic into a general set of a definition, which is not platform or language dependent. Based on this definition, one can automatically create the programs based on this definition, without the need for manually include more logic in the generated source, so the generation of the programs can be repeated any time for any language or platform.

The goal – to be honest – is to have a tool for creating 90% of the programs needed for business apps. Primary target platform is IBM I – but you can use the definition to create programs for any platform / language. Maybe even multi-language should be possible later…

The remaining 10% of the application? Well – this is to be discussed, or can be solved individually.

What “XTend” is not

In its current state – and with current completeness of definition – there does not exist a tool which will help you in any situation and will allow you to stop programming. This is a goal of this project, but we are lightyears away from that. But we have this idea…

Currently, there is no definition for representing logic in general, besides some input validation or database reference checking. Currently, there is only a designer program for list and work screens which runs as a Windows32-program. Currently there is only a generator for IBMs outdated macro language “Net.Data” to create some web programs with CSS. But when you read on, you will see the idea behind the project, you might have potentially more new ideas and you might be interested to be part of the project…

Alternative ideas

If you think there already is a tool or set of documents (yes, I know of modeling languages) which can replace this project – just drop me a note ;-) The current state, the set of definition and the programs do work for *me* at the moment, but it should be modernized and extended. The definition is neither fixed nor state of the art, so every idea, discussion or critics is welcome.

Definitions and terms

Let me first introduce you into some details of the currently existing project and outline my ideas for the future. Not sure, if all of these ideas can and will be realized, but this mostly depends on you (ie – how many people will use and extend this project).

One main topic is: *the definition* is the most important part of XTend, and it is open for everyone. If you read the definition documentation and understand the concepts, you can make your own program generator which can create blazing apps based on the definition in the language of your choice!

So there is three major parts this project consists of:

The “definition”

As you just learned, the most important part of this project is “The definition”. This is a set of files which contains information about everything which makes a business application:

- database entities (files) and fields (attributes)
- relations between database entities (depending on database definition)
- references from one file to another file (for example order master file and customer master file, linked by the customer number)
- validation rules for fields in the database or on a screen (for example a specific numeric field only may contain values between zero and 99)
- screens of different types like “list” (eg. list of customer master data, list of orders) and “work” (work with customer master detail data) – maybe more types...
- fields used on a screen with details on how to display them (a list screen contains several columns, a work screen contains some fields from the master file) as well as how to display them, is a user allowed to search in a list field, what data is allowed in a work field etc. (this may get rather complex)
- additional info for all fields on a screen (validation checks, external references for example to grab customer name for a customer number)
- linking between screens (e.g. a click on customer number opens work with customer master data screen), a definition of a process to display several screens in a given order etc.
- etc… what ever you want

Main goal of "*the definition*" is to combine database entities and logic rules which make up your application without any platform or language specific details. A validator is a set of rules with allowed values for a field, or a reference to a table containing allowed values. A field can be a simple text field, or a date field, or a yes/no checkbox etc. All of this can be entered into the definition and later be realized in any language. So we are rather platform independent. There is only one exception: you will be able to add SQL rules which can make life more easy for the people writing the generator…

As "the definition" is the base for all other parts of the project, it is the most important part and should be maintained and extended in a very documented way. Then there is a method of being independent.

The "designer"

As all contents, data, values, references which form "The definition" is located in some database tables, you might use the database tool of your choice to create this data, but this is not very funny and needs a lot of time – and will result in errors. So there is a need for a tool to *design* screens, or to enter data into the definition tables. This is we call "*The Designer*". This is (currently) a Win32 program which allows you to:

- define projects (for sure you will have multiple projects)
- define the main database for the project. You can use ODBC to link to an IBM I library and import information about all database tables and fields, so you do not need to manually enter this in your definition. This is no database design tool! There are millions of them. Use your favorite method to create your SQL database in an IBM i1 library and then import to the designer. If you add fields, import again (it is not recommended to delete fields at this stage of the project).
- define validation checkers which you later can use in your screen definitions to be sure the user only enters valid data
- define external references you can use later to get detail data in your screens based of a database field value (eg. customer number > customer text)
- define screens. This can be a simple list of customers where you can display the most important fields and allow some fields to be sorted or to be searched in. This also can be a work screen which allows the user to view / edit some or all fields of a database table. The look and feel of the screens is only roughly defined by listing them. The real user interface will be created later and depends on the capabilities of the creator used for making the sources. On a small mobile phone, a work screen for sure will look differently as on a normal PC screen. There should be means added to "The Definition" to reflect different display types...
- more to come. The more "the definition" set grows, the more possibilities should be included in "the designer" and "the creator".

The “generator”

As soon as you have a set of definitions (even a single file and a single screen is a good definition), “the generator” will come into the game. Based on the “definition”, now a real program must be created.

As the definition does not contain any system or language specific parts (except for SQL, see above), you can create your own generator to create source codes (or even executable programs) in any language for any system, which will then access data, display data, verify data, write data, and (somewhen) execute some more complex logic. The “generator” can read the definition and write text files which make a source code, the source then can be compiled.

Maybe you are keen and directly create a Windows executable, or create java classes, or create SQL/RPG sources for a green screen, or want to realize everything with PHP or node.js. It’s up to you!

You also might want to create some sources which will then be used to compile apps for mobile phones. This should be possible if you keep in mind to create the user interface based on the “definition” and consider possibilities of the target platform.

The “generator” will be the most complex part of the whole project, as teams will be needed to create a generator for the platform / language of their choice. Maybe competing with other teams for the same or different language. Competition can be progress :) Maybe you also create an “Interpreter” instead of a “generator” to dynamically create displays and execute logic based on “the definition”.

The “generator” also should include some deployment tools if deployment is more complex than just copy some files. Maybe you simply want to copy to GitHub, or FTP your PHP scripts to a web server...

Also to be considered in your generator is a lot of testing, because you possibly will create hundreds of programs based on the “definition”, so you must check against all possible errors or security flaws. But on the other hand – if your generated code works for one generated program, all other generated programs will function the same and have the same level of design security.

And in the end – there never is an end in creating a generator. If there are new features included in the “definition”, all generators should be updated to reflect this change.

Current state of the project

Parts of this idea have started shortly before year 2000, so: this is an *old* project which has been sleeping some time. It has been grabbed out of the drawer in my desk some three years ago as a customer had an idea for modernizing their green screen based ERP software. But the did not want to simply “modernize” or rewrite the application or get a new software. They are glad with the programs they have, they just wanted some additions, and maybe some easy data entry programs for new people not used to pressing F-keys on a text based user interface. So the goal was to create web based programs working on the same database with almost same functionality. As there was not much validation in the database (like constraints) and the data entry programs contained a lot of individual logic, an analyzation has been started to see what can be “defined” without thinking too much in RPG or a language in general.

Consider this situation: a new coworker should update the customer master database, edit some fields, update the phone numbers, or verify if a customer has ordered for the last months and maybe someone should give the customer a call. The new coworker is 20 years old and never used a green screen. They love the mouse and only know about web based applications.

So what to do? Train them for using a 5250 based application? Hmm… rewrite all programs dealing with the customer master database for browser access and train the “old” people who are used to work on green screen at blazing speed without using the mouse? Nah… So let’s give both of them the tool they need: the existing users use existing programs. New users get new programs (maybe with a limited set of functionality first). So this path of modernization will include:

- analyze logics in data entry screens (5250)
- define and rebuild with XTend
- freeze change to 5250 programs if there is no plan to update RPG programs
- slightly move users to newly created programs.
- If there is a generator for 5250 programs, why not still create them for people being used to that old-fashion interface whilst having modern programs for the others.

As we have an existing database (which does the job and contains all data needed), we do not fiddle around with it. Don’t do so! Just create a program to read and write.

So import the database definition from the production data library. Create a list screen with some 5 important fields (customer number, customer name, telephone number etc). Create some work screen to view (and maybe edit) important detail fields. This can be done with XTend within 10 minutes! How? Read on…

The existing definition

The existing “definition” holds some attributes for every field in the real database. Most of them deal with how to display data, and how to verify users input. A description of the definition (files and fields) will follow later in this document (see the attachments). This document is not yet finished, to be precise, it has just been started. So there is a lot of information missing which will leave questions in your mind. Don’t hesitate to ask, the answers will be added. If I do not answer your email, please wait for an update of this document.

The existing attributes are ok for defining:

- list screens (a set of columns in a table to display, including data references, sorting, searching) which display multiple database rows
- work screens (a set of columns in a table to display, allow user to edit, validate, and save to the database) which displays a single database row
- for both screen types: attributes like readonly, uppercase, check against a table based on key values etc. These attributes try to represent all “easy” types of logic which usually are hard-coded in an old-style program
- allow the user to click on a key value in a list screen to open a work screen (for display or edit or delete).

Not yet realized but planned:

- add possibility for very basic logics like “IF field value=A then set other field=B” etc
- menu definition and menu display (for the whole application)
- security based rules for restrict user access to screens
- more attributes
- more complex logics
- external calls
- create APIs for each screen if wanted to realize micro services
- etc…

The existing designer

The designer is a Win32 application written in Delphi 5 (yes, that old), because parts of this projects – or at least basic routines – were started in the late 1990s where D5 was a current development environment. But – wow – it still can create programs which run on a windows 10 box. Maybe I find time to compile using Lazarus to run the designer on my mac :-) Or if you have some time, create a designer for web browser use in a language you prefer. The designer does its job, maybe has some little flaws and does not conform to current requirements to a GUI – but that's the reason I am looking for enthusiasts. Maybe some will do the designer job…

The current designer executable also includes the "generator" (one big binary).

Details on using the designer will be included in this document as soon as there is a distribution available.

The existing generator

The currently existing generator is being developed in parallel to the designer, as both parts are realized in the same set of programs / executable and whilst the list of attributes in "the definition" is being extended. This "big binary" is not needed and might be changed later – but at the moment it works – for me.

The generator also consists of old code and only creates IBM Net.Data – a language IBM dropped 10 years ago. So it is time to create a new generator – maybe a good start to learn new languages like node.js or whatever is suitable to create web apps.

How to test?

At the moment, development of designer and generator as well as the usage for development of real projects is used on my PC. If you are interested, it is possible you can get the binaries (and Delphi 5 sources, if you dare to read them) for your PC. But at the moment, you need an IBM I system to test the generator results. The definition database can reside on any ODBC database, but I am afraid there is no Windows executable for Net.Data scripts (any longer…)

So this will be of not much use for you at the moment unless you got an AS/400 or IBM I with OS/400 V4R3 or higher (whatever runs Net.Data).

It is planned to create some environment on pub400.com (see http://pub400.com) so users would be able to have an own environment there, but this needs some time as setup will be a manual process. If you really (!) are interested and plan to invest time for this project, please drop me a note so I can setup things for you...

How to participate?

The current restrictions stated above urge that any enthusiast wanting and willing to take part in pushing this project forward should have the following resources:

- time (!)
- ability to read Delphi5 code (code base for Designer and Generator) – if you want to understand the way the current generator works.
- ability to read Net.data code (generated code, but Net.Data is rather easy and fast to learn compared to PHP etc.)
- have an IBM I where you can store some data with ODBC and where you can setup a web server instance to execute Net.Data scripts (we can be of help to describe what todo)
- You should agree to not receive a ready-to-run software for creating your projects. Be keen and play around until you get your result.
- You should not expect a finished documentation. This is a topic just started…
- You should be willing to test and report and to deliver ideas (a discussion forum is to be created. Any ideas?)
-

Documentation

This is the description of the most important part of this project: The definition, which consists of some files containing information about entities, relations, programs, fields and attributes. Please keep in mind:

- The names of files and fields described here might change to reflect a proper project naming (base was “NETDATA” years before IBM abandoned this small and quick script language). Before we are going to extend the definition, this naming scheme will be cleaned up. Currently, most files and field names start with “ND” and I consider changing this to “XT”.
- Some existing attributes and definition ideas are just ideas and not even realized in the current generator code. Sometimes – in a creative moment – I just have an idea, add the attribute and later realize the code in the generator to see if it is a good idea or just fancy stuff. Maybe it can be of use for someone later.
- Nothing is perfect, so is this project. And there is a lot of ideas missing so don’t be afraid to bring in your ideas – even the ones you might find rather crazy.
- The definition of fields in this database should be discussed. Some fields are rather small (for example ID fields). Thanks to SQL it should not be a big issue to increase length of the fields. One of the next planned steps is to do this to reflect the possibility for creating bigger projects before starting the work on expansion[1] of the definition’s capabilities. For most of the fields there does exist one urgent rule: The length of the fields will be increased...
- Also the code (if you want to read it) is not perfectly written. Sometimes copied elements just to get it running. Sometimes no documentation at all. There is so much room for improvement...
- All files reside in an SQL schema (or: Library) – if you want to adopt this to a different database, just keep them together and keep the naming scheme.
- “symbol name” means an ID used by the executing system, like user name, library name, schema name, object ID etc. Must be uppercase and may not contain special characters like blanks, umlauts, quotation marks etc.

1 The term “expansion” refers to working on the definition set by adding fields to reflect new attributes or to add files for new functionality.

File “NDPROJECT”

Note: Planned new name: XTPROJECT

This file contains information about the projects you work on. For sure you can work with multiple projects in your XTend environment. Some of the fields here are currently not used in the generator but just have been added for future use.

NDPROJECT fields

- **PRJID** (Alpha 10) – a unique ID for your project. Must be uppercase without blanks. Will be used as key field in all other files. It is planned to have at least one special project ID named “*SYSTEM” to combine definitions for screens, fields or programs which can be used in all projects without defining them multiple times.
- **PRJDESC** (Alpha 80) – a free text for you to describe the project.
- **PRJDTALIB** (Alpha 10) – a schema name for the primary database scheme of your project. In a later stage, you project should be able to use multiple schemes (for each user group etc), this is to be done by definition expansion.
- **PRJMACMAIN** (Alpha 40) – The name of the main (or starting) script for your generated source codes.
- **PRJPATH** (Alpha 40) – A path name where the generated source should be saved to. This is mainly of importance for a deployment module which does not exist yet.
- **PRJIMG** (Alpha 128) – A path and file name for an image which is to be added on your generated application (if not done by CSS or whatever).
- **PRJGLOB1** .. **PRJGLOB5** (Alpha 40) – Five fields which can include a symbol name for a variable each. These variables can be used in the generated scripts and should be regarded “global” including handover from one program to the next one (if there is a link or process). This can be of use in a kind of “session” management where the user selects some globally valid value like sub-company etc. [to be discussed]
- **PRJSERVER** (Alpha 128) – a name (DNS) or IP of the server for sending the generated code to. Planned to be used for deployment.
- **PRJCTLLIB** (Alpha 10) – a schema name where the generated code can save control data like session management, or access management data, if your generated programs will use multiple data libraries. [to be discussed]
- **PRJGDECCOM** (Alpha 1) – boolean value (0 or 1) to tell the generator if globally all decimal values will use a comma (1) or point (0) for displaying or accepting decimal values. At a later stage of this project, country code dependent settings should be used instead of this value.

- **PRJGDECZERO** (Alpha 1) – boolean value (0 or 1) to tell the generator if globally all decimal values will have a leading zero (1) or not (0) left of the decimal comma/point. Should be replaced by country code settings later.
- **PRJLANG** (Alpha 3) – code for language to be used in your generated programs if text constants are not used from a separate database table. Currently only ENG (for english) is allowed and not used in all parts of the generator.
- **PRJADMIN** (Alpha 40) – a symbol name stating a special user name which will have all access in your generated programs. Mainly used for debugging at the moment. In a later stage, a complete rights management within the generated programs (in combination with a menu function) is to be realized.
- **PRJAUDIT** (Alpha 1) – a boolean value (0 or 1) which tells the generator to create database files and all code for a full audit trail. This is useful when your generated apps run in a regulated environment. Some tables will be created to receive information about: when a user opens a program, when data is read, modified or deleted. Each file access program created by the generator will include the appropriate logging routines.
- **PRJPGSIZW** (integer) – an integer value (0 .. 100) which is currently being used for creating list screen programs. A maximum of $PRJPGSIZW lines is displayed, if the list contains more lines, navigation elements and paging will be created.

File “NDTABLES”

Note: Planned new name: XTTABLES

This file contains information about the database files you will use in your projects. Usually, contents of this file will be created using an import function (existing in current designer), which will read a given SQL scheme and import all existing files and their field attributes. If you re-import your files (for example after you added fields), most of the data in this file will be replaced!

NDTABLES fields

- **NDTPRJID** (Alpha 10) – The project this table is used in. Usually, for a given project you have one database (or maybe multiple databases with the same files and fields). If you want to use a given database or table in multiple projects, you must import its description for each project.
- **NDTTABID** (Alpha 128) – The name of a table. As we can use SQL naming, no old DDS-Naming from AS/400 (10 chars) is used. But luckily, each AS/400 erm IBM I physical file has also a long SQL name, even if it is the same short one ;-)
- **NDTTYPE** (Alpha 1) – For future use. Currently this field always contains “T”.
- **NDTTABDESC** (Alpha 80) – The description of this table as reported via SQL or ODBC. So it is a good idea to correctly comment on your tables before import.
- **NDTSNAME** (Alpha 10) – Short IBM name of a database table. Not used any longer, so mostly empty
- **NDTREMARKS** (Alpha 2000) – Planned to be used for additional comments entered in the designer. Currently set to the same value as NDTTABDESC when tables are imported. *It is planned to remove this field from this table!*
- **NDTSQLTYPE** (Alpha 24) – A literal which shows the SQL type of the file. This can be either “TABLE” or “VIEW”.
- **NDTHASPK** (Alpha 1) – If the import routine finds a unique key for the imported SQL-Table, this field will be set to “1”, otherwise to “0”. It will be used in many parts of designer or generator to decide, if this table can be used as entity table, and if we can create an API in the future with unique access.

As you see, this table uses a rather small set of attributes, as they usually do not change whilst you design a project. There is a second place to store additional information about a table which is being used and shall not overwritten when a database re-import is run.

File “NDTABLES2”

Note: Planned new name: XTTABLES2

This file contains additional information about a table which will not be overwritten when you re-import an SQL schema.

NDTABLES2 fields

- **NDT2PRJID** (Alpha 10) – The same project ID as in all other files ;-)
- **NDT2TABID** (Alpha 128) – The table name as in NDTABLES. These two fields will link to that file.
- **NDT2HIDDEN** (Alpha 1) – If set to “1”, this table will not be shown in the designer when you can select a table. Usually, if you import a Library which contains non-database files like QCLSRC etc, you set them to “hidden”.
- **NDT2COMM** (Alpha 1024) – you can comment your table here. This will be kept even after re-import. It is planned that this field replaces the NDTREMARKS field.

There will be more fields to come…

File “NDUKEYS”

Note: Planned new name: XTUKEYS

This file contains information about unique keys for imported tables. It is used to quickly select key fields in the designer or generator.

NDUKEYS fields

- **NDUPRJID** (Alpha 10) – The usual project ID
- **NDUTABID** (Alpha 128) – The table ID
- **NDUORD** (Integer) – the ordinal number of this key field.
- **NDUCOLID** (Alpha 128) – The name of the key field

If the designer or generator needs to know about the unique key of a file, the records here will be read ordered by NDUORD. For generated SQL it might be possible to use the key fields in any order, but some database optimizers work better if the key fields are ordered in the way they have been defined first.

(you will see that this information is some redundant, but I’d like to have performance in the generator).

File “NDCOLUMNS

Note: Planned new name: XTCOLUMNS

This file contains information on all database fields available. This data mainly comes from the database import routine.

ND COLUMNS fields

- **NDTPRJID** (Alpha 10) – The well known project ID
- **NDCTABID** (Alpha 128) – The long table name
- **NDCCOLID** (Alpha 128) – The long column name (we will not use the short system field name with 10 characters)
- **NDCCOLTYP** (numeric 4) – A numeric representation of the column type as reported by the DB2 database. This might be incompatible with other databases, so a different solution is evaluated. Valid values can be reviewed at: https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/db2/rbafzcatsqlcolumns.htm (see description for DATA_TYPE).
- **NDCCOLLEN** (integer) – The maximum length of data in this column. Will be used for all validity checking and automatic field definitions.
- **NDCCOLDEC** (integer) – If the column is a numeric data column, this field contains the number of positions after the decimal separator.
- **NDCCOLNUL** (Alpha 1) – If this column may contain NULL values, this fields contains “1”, else “0”.
- **NDCCOLDESC** (Alpha 80) – A short description or heading of the column as imported from the database.
- **NDCKEYSEQ** (integer) – If this column is part of a key, then this fields shows the numeric sequence of the column in the key definition, else zero.
- **NDCORDPOS** (integer) – this shows the position of the column in the table as in the same sequence as defined.

As you see there maybe is more information which should be retrieved from the database, but at the moment the fields above are ok for the projects goal. There may be a second file which contains information for each column which is to be kept after database re-import.

(Plan: NDCOLUMNS2)

File “NDSCREENS”

Note: Planned new name: XTSCREENS

Now the fun begins :) This file contains a list of all screens defined for your projects. This also contains some attributes; some of them may be experimental but can be of good use in your programs, depending on the menu or process structure. Just read on...

NDSCREENS fields

- **NDSPRJID** (Alpha 10) – The project ID this screen is defined for. The special value of “`*ADMIN`” is currently being used for testing general programs which will be created for all projects, mainly for user, menu and security management.
- **NDSSCRID** (Alpha 128) – The name of the screen definition. Although this name can be very long, it is recommended to use short names. If you are going to use XTend to rebuild an existing 5250 application, you maybe want to use the short RPG or DDS file names for the screen ID.
- **NDSSCRTYPE** (Alpha 10) – This field shows the screen type. Currently, only two values allowed: “`LIST`” for list screens, and “`WORK`” for work screens.
- **NDSTITLE** (Alpha 80) – This text will be displayed on selection list in the designer as well as the title of the generated program.[2]
- **NDSEDITOR** (Alpha 20) – This field will contain the user name of the person currently working on the screen definition. If non-empty, your designer program should not allow editing of the screen. After saving the screen definition, this field should be empty again.
- **NDSPRITAB** (Alpha 128) – The primary table for the screen. Screens usually have a primary table, like LIST-screens which display rows from the primary table, or WORK-screens which read a given (by key) record from the primary table. *To be discussed is the use of an SQL statement as primary table in future versions.*
- **NDSSINPK1** .. **NDSSINPK5** (Alpha 128) – These are five (5) fields. They can contain a column name each for a primary input key. A primary input key is used when the defined screen is called from an other screen and data must be read by a key, so these key fields will be used with the parameters received by the screen defined. For example: you have a list screen called “LCUSTOMER” which displays the customer file. One column is “CUSTNO” which is the customer number. You want to open a work screen “WCUSTOMER” when the user clicks on the customer number. So “WCUSTOMER” has the customer file as primary table, and NDSSINPK1 is the column name “CUSTNO”. The work screen then will use this column to read the data given in the first input parameter. So you can call “WCUSTOMER” from any screen which sends one parameter. Ok, you may take the article number in a display “LARTICLES” and call “WCUSTOMER” if the article

2 currently, lots of strings are not flexible defined to allow multiple languages. This needs to be changed or added later.

number column also is numeric, but that does not make sense. But from a technical view it will work. In the designer, each list screen can have up to five data fields which will be sent to an other screen which uses them as input primary key fields. The designer roughly compares the field types. A practical example will follow in the designer documentation. **NOTE:** This principle with a maximum of 5 input parameters (and output parameters in list screens) is a first version of a definition. In future versions this should be put into a normalized version using an own definition file…

- **NDSGENADD** (Alpha 1) – If set to "`1`", the generator will create an "add" button or function in a list screen so the user can add a record. See next column. *(LIST screens only)*
- **NDSGENADDSCR** (Alpha 128) – Here you define the screen name of a work screen which will be called if NDSGENADD is set to 1. *(LIST screens only)*
- **NDSGENEDIT** (Alpha 1) – If set to "`1`", the generator will create an "Edit" button in front of every data row of a list screen. The generator will use the primary key (maximum 5 fields) of the primary file when calling the work screen. *(LIST screens only)*
- **NDSGENVIEW** (Alpha 1) – If set to "`1`", the generator will create a "View" button in front of every data row of a list screen. The generator will use the primary key (maximum 5 fields) of the primary file when calling the work screen. *(LIST screens only)*
- **NDSGENDEL** (Alpha 1) – If set to "`1`", the generator will create a "Delete" button in front of every data row of a list screen. The generator will use the primary key (maximum 5 fields) of the primary file when calling the work screen.[3] *(LIST screens only)*
- **NDSWHERE** (Alpha 300) – Here you can add a general SQL clause which will be added after the SQL text to select data, so you can filter your data on a list screen. For example you have a customer file with field CUSTAT which contains "`D`" if the customer account is inactive. Now you can set NDSWHERE to "`CUSTAT <> 'D'`" and your list screen will not show any inactive customers. (Take care – no double quotes needed). The generator will at this String to any WHERE clause used in the list screen – even when you have 10 search fields.
- **NDSDEL4UPD** (Alpha 1) – Now it's getting some complex. This field can be used if there is some weak database design, and your primary file for a work screen does not have a unique key. Updating records might create new records if no full key is specified. So you can use this attribute (set to "`1`") as a workaround, the generator will create a DELETE then an INSERT statement. Use should be avoided, update your database design! *(WORK screens only)*
- **NDSHINT** (Alpha 1024) – Here you can define an unformatted text which can be displayed on the LIST or WORK screen for user guidance. Depending on how you

3 the EDIT, VIEW, DEL buttons created rely on a unique key. The designer will check the key definitions.

create your GUI, you might display this text at the bottom of the screen, or using a special Help button or the F1 Key.

… much more to come …

As this file will grow, it is planned to “normalize” the database design into a flat attributes table to make extensions easier.

File “NDSCRCOL”

Note: Planned new name: XTSCRCOL

If you considered the Screen-Definition file is complex, have a look at this file defining the columns on a screen. As this file serves for WORK screens as well as for LIST screens, a lot of attributes are valid only for one type of screens. Some of them are experimental and might be removed later[4].

NDSCRCOL fields

- **NDSCPRJID** (Alpha 10) – The Project ID
- **NDSCSCRID** (Alpha 128) – The Screen ID (see above)
- **NDSCCOLID** (Alpha 128) – The column ID (as in the primary database file)
- **NDSCWIDTH** (Alpha 10) – The display with on screen. This field is *not* numeric to allow html table style values like “25%” or “*”. Usually on list fields you have some small columns (use absolute pixels) and one or two large columns (use “*” at least for one column), so if the generator creates a table, the browser can adjust the display. *(LIST screens only)*
- **NDSCHEAD** (Alpha 128) – On database import, the column heading is retrieved from the field description. This might be unwanted if the text is too long for a small column. So you can here overwrite the column heading. *(LIST screens only)*
- **NDSCSORT** (Alpha 1) – Enter “1” if this column can be sortable. If activated, the generator will create buttons for sorting the table by this column ascending or descending. *(LIST screens only)*
- **NDSCSEARCH** (Alpha 1) – Enter “1” if the user should be able to search in the contents of the column. Depending on the field type, there might be different search types, see field NDSCSCHTYP. *(LIST screens only)*
- **NDSCHIDDEN** (Alpha 1) – Enter “1” if this field is not being shown on a screen (although its contents might be used later in a logic block)
- **NDSCLINKTO** (Alpha 128) – Enter a screen ID here to enable the user to click on the value of this field in a list screen. The generator will create a link (a href) or other means so the given screen is called, using the field value as parameter sent to the target screen (which should have at least one input parameter). Usually you have a LIST screen linking to a WORK screen, but a LIST screen as target is also possible. *(LIST screens only)*
- **NDSCLINKTYPE** (Alpha 1) – A Link to a target screen should be either “E” (edit) or “V” (view). This attribute might be overwritten later by other attributes.

4 as written before, we try to not delete fields to be compatible. You can decide in your generator to support deprecated attributes.

- **NDSCLINKHINT** (Alpha 80) – Use this attribute to display a text for the user as hint for the target program. For example you might want to have a mouseover displaying "Edit this customer" on a customer number which links to the appropriate WORK screen.
- **NDSCLINKOPT** (Alpha 60) – This field defines, if a LINKTO shall open a new window or not. It currently supports a maximum of 60 fields with a LINKTO definition in a LIST screen (which should be enough). For each field on the screen one digit in this string will be either "`1`" or "`0`". So if your LIST screen has 6 fields, where the second field has a LINKTO which should be open in a new window, this string will read "`010000`". The design of this field might be worth to discuss :)
- **NDSCPOS** (Numeric 5) – The fields on a screen are displayed in the given order. Depending on the creator and the target device this is useful if you only can display one field per line.
- **NDSCSCHTYP** (Alpha 1) – If the LIST screen column is searchable (depending on field NDSCSEARCH), this attribute defines how the column contents can be searched. Allowed values:
  - "`L`" (Like) – the data will be compared using SQL LIKE "%searchstring%" clause, so any data containing the entered string will be searched. It is advised to remove percent signs (%) from the search string in the generator.
  - "`E`" (equal) – the data field contents must be equal to the search value. The generator will create code depending of the field data type to compare numeric or alphanumeric.
  - "`G`" (greater equal) – the data field contents must be equal or greater to the search value. The generator will create code depending on the field data type.
  - "`C`" (content list) – this will have the generator create a list of distinct values to be selected by the user. It is advised to use this attribute only with fields where the number of distinct values is rather low (<100), else the GUI might produce a very large select box.
- **NDSALIGN** (Alpha 1) – How to align the display of field data in a LIST screen. Allowed values are: "`L`" left oriented, "`C`" centered, "`R`" right aligned. Default value in the definition is "`C`".
- **NDSCOP1** .. **NDSCOP5** (Alpha 5) – This are five (5) fields where you can define up to five field names. The content of each field will be used as output parameter if you have defined a link (see NDSCLINKTO). Usually, a link is being created with a simple unique key consisting of a literal or a number. But in some cases it is wanted to use a key consisting of multiple fields. So you can create the link on one column field and handover additional field values needed by the target program to identify and load the record. This definition allows for a lot of possibilities! Usually, the NDSCOP1 is the same field name as NDSCCOLID (the list column we want to add the link to). But imagine you do not want to display the customer number in the list

but only the customer name. So define NDSCLINKTO on the customer name field, and hand over customer number in NDSCOP1 (which for sure must be read by the target screen).

- **NDSCTYPE** (Alpha 1) – Type of contents. If the exact type of contents cannot be derived from the imported data type, you can specify the type here. The generator will then handle the output accordingly. Allowed values (not all might be suitable for each display architecture):
  - "T" (Text) – field data will be displayed as normal output text
  - "M" (Mail) – this field contains eMail addresses. If a content is to be displayed, create a mailto link so the user directly can click the field data to send an email.
  - "I" (Image) – this field contains an http Image URI so the generator will create an appropriate <img> tag to display the image at this URI.
  - "L" (Link) – this field contains an http URI. Generator will create a clickable link
  - "C" (Checkbox) – Field will be displayed as a checkbox (yes/no). To define the database values representing "yes" or "no", see below NDSCYES / NDSCNO field definition.
- **NDSCHINT** (Alpha 1024) – this text can be used as additional help for the user regarding a particular column. Depending on Screen type (LIST, WORK) and display architecture, the generator can create a mousover-textt, or user can press a Help button or F1 key.
- **NDSCFLDX** (Numeric 5) – on a work screen, you can define a numeric column value here. Depending on the screen display architecture, you can control the starting column on a WORK screen to display fields contents. *(WORK screen only, planned for 5250 generator).* Recommended Values: 0 .. 131
- **NDSCFLDY** (Numeric 5) – on a work screen, you can use the designer to position fields on a row/column base. Here you can define the line number where to display the field data. Currently, the generator only creates web based work screens and display fields line by line based on NDSCPOS. Fields NDSCFLDX and NDSCFLDY are planned for future use on a green screen. *(WORK screen only, planned for 5250 generator)*. Recommended Values: 0 .. 26
- **NDSCDSCX** (Numeric 5) – if the field description is to be displayed on a 5250 screen, this gives the start column number. Usually, you display the field description on the left side of a screen, followed by field data and possible detail data. *(WORK screen only, planned for 5250 generator).* Recommended Values: 0 .. 131
- **NDSCDSCY** (Numeric 5) – if the field description is to be displayed on a 5250 screen, this gives the start line number, so for example you can display the description above the field data. *(WORK screen only, planned for 5250 generator)*. Recommended Values: 0 .. 26

- **NDSCFLDW** (Numeric 5) – this defines the width of the entry field on a WORK screen. For a long text field, you can define a smaller edit field size to not ruin your web browser GUI design :-) Generator should be extended to automatically create text-area fields for larger text fields (or do you plan to edit data in a 1000 character field in a single line?)
- **NDSCMANDAT** (Alpha 1) – if set to "1", user **must** enter data into this field. *(WORK screen only)*
- **NDSCRO** (Alpha 1) – If set to "`1`", the WORK screen field always is read only.
- **NDSCUPPER** (Alpha 1) – If set to "`1`", the WORK screen field will always convert user input to UPPERcase character.
- **NDSCYES** (Alpha 10) – If the field type is set to "Checkbox" (see NDSCTYPE), you must define which database value does represent "yes" and which does represent "no". In most applications, this will be done using "1" and "0", but sometimes this also can be "Y" and "N" or whatever. If your database is some talkative, you can define a string up to 10 characters. The value can be case sensitive, so please check your application data first! This value is sent to the database from a work screen when a Checkbox field is checked and the user saves the data.
- **NDSCNO** (Alpha 10) – The "no" value for a Checkbox field. This value is sent to the database for an unchecked field.
- **NDSCYES2** (Alpha 10) – Sometimes it is needed to have an alternative "yes" value if your database was changed without proper data update. So the generator sets a checkbox to "checked" if NDSCYES or NDSCYES2 is found in the database. At database update, the value of NDSCYES is sent, so NDSCYES2 is only an alternative value for display. If these two fields have different values, this may result in data change when the user saves data on a work screen. You have been warned. If you leave this field empty, no alternative "yes" display value is used.
- **NDSCNO2** (Alpha 10) – The same logic as NDSCYES2, but for "no" value.
- **NDSCHIDVAL** (Alpha 1) – If the database value of a checkbox field shall not be displayed, set this to "`1`". Otherwise, the generator will show the current database value right of the check box.
- **NDSCNOBLANKS** (Alpha 1) – If set to "`1`", the WORK screen edit field may not contain blanks (spaces).
- **NDSCDFTNEW** (Alpha 40) – If any data is entered, this will be sent to an empty entry field on a WORK screen. This field is untyped, so do not enter alpha data for a numeric field!

- **NDSCCONT** (Alpha 1) – Specify the content validation of an edit field on a WORK screen. The following values are allowed, although only “N” is realized in the current generator program:
  - “`*`” or (blank) – default, try to guess allowed content from the database definition
  - “`T`” – All text / any data is allowed
  - “`A`” – Alphabetical chars only (a..z, A..Z)
  - “`N`” – numeric only (real numbers including decimals)
  - “`D`” – Date (date format not yet specified)
  - “`Z`” – Time (time format not yet specified)
  - “`S`” – Time stamp (SQL DB2 standard)
  - “`R`” – integer, numeric range (further definition needed)
- **NDSCHIGHL** (Alpha 1) – If set to “`1`”, the field data is highlighted on the screen (depending on the displays capabilities)
- **NDSCVALVAL** (Alpha 40) – Going to be complex. You can define a so called “validator” (see later) which defines a set of allowed values or rules in general. For example, a validator can define a list of valid entries, or a valid numeric range from 0 to 50. Here you define the name of a validator, if you want to have this fields data checked against such a rule. If empty, no validation is done.
- **NDSCVALPRJ** (Alpha 10) – Enter the project ID of the validator named in NDSCVALVAL. You can have validator definitions unique for each project, or available for all projects (which is good for general validators like “byte” which is a range from 0 to 255). It is possible to use a validator name multiple times (for several projects or for all projects, but this can be confusing. If a validator is available in all projects, NDSCVALPRJ must be set to “`*ALLPRJ`”.

Now it is going to be really complex. The following fields (still in file NDSCRCOL) deal with a so called “reference”. In general, a reference is a definition of a table which contains detail information for a key value. For example, you have a customer master file containing all information on your customers. The key field to this table is the customer number. Now you have a primary table in your LIST or WORK screen which includes the customer number only, but you want to display the customer name or city. An invoice master file can be a good example. So you must access the customer master file with key “customer number” to get the customer name as an output (we name it the “detail” information).

This is when a reference will be of help. The reference defines at least a table (the customer master file) and the reference field (the customer number field) as well as the output field (the customer name). If this reference is attached to a field in your primary file, the generator creates the code needed to get the customer name and return it to your LIST or WORK screen which then displays the name. As the customer number is numeric, you may attach this reference definition to any numeric field (e.g. invoice number), but then your application will display unwanted data :-)

You can define all needed reference data for each single field in your screens, or you can create a reference definition (see below for the file NDFREFS) so you can reuse a reference definition at multiple places in your application and only need to create the definition for this reference once. And later you only change the reference definition (maybe to add a second output field with more customer name information) and you’re done for all situations where you have a customer number...

As a reference can be a lot more than just a single input / output field, there is much more parameters, so lets start with the easy ones. Our example deals with customer number pointing to the customer master file to get the customer name. So we define the following fields on a LIST screen with invoices to get detail info on the customer number.

- **NDSCREFTBL** (Alpha 128) – Enter the database name of the external table where you want to get the detail information from (e.g. customer master file).
- **NDSCREFFLD** (Alpha 128) – The name of the reference field (which you want to compare the data of the current database field with). This for example is the customer number.
- **NDSCREFOUT** (Alpha 128) – The name of the field in the external table, which contains the detail info we want to output. This is the field “customer name1”.

With these three definition fields we are basically done to get our information. Now come a lot of more fields, as a reference can do much more for you…

More reference fields:

- **NDSCREFOUT2** .. **NDSCREFOUT5** (Alpha 128) – If you are interested to get more detailed information from the external table, you can add up to 4 more field names into these definition fields. Maybe you also want to have returned the street, ZIP code and City, and the telephone number (a lot of data to display, pay attention!) The generator code will grab these fields data and return them in a single string.
- **NDSCREFCON** (Alpha 10) – If you define multiple fields as detail output, you can define how their contents are concatenated and returned. Maybe you want to have them separated by a space (you also can enter " " to have a fixed HTML space), or you want them separated by a comma. Feel free to enter up to ten characters for concatenation.
- *NDSCRCOL2 / NDSCRCOL3 – Not yet defined*
- *NDSCRUP2 / NDSCRUP3 – Not yet defined*
- **NDSCDETAIL** (Alpha 1) – If set to "`1`", the defined reference will be used to display the detail data. You can set this to "`0`" to NOT display the detail data without deleting the reference definition (to use it later again).
- **NDSCREFSQL** (Alpha 1024) – You can add a constant which will be added to the SQL WHERE clause by the generator. This can be of help, when you need more selection in your reference. Maybe you just want to get a detail data from your customer, if the customer is not set to "inactive" and in your customer master file this is represented by an "`A`" in field "CSTATUS". So enter here "`CSTATUS='A'`". The generator will do the rest...
- **NDSCDETX** (Num 5) – (Planned for 5250 and web) Display the detail data at a special column on your screen, either left or right of the field data. *This is not yet realized in the current generator!*
- **NDSCDETY** (Num 5) – (Planned for 5250 and web) Display the detail data at a special row on your screen, either left or right of the field data. *This is not yet realized in the current generator!*
- **NDSCDETL** (Num 5) – (Planned for 5250 and web) restrict the output of detail data to a given number of characters. *This is not yet realized in the current generator!*

- **NDSCLENGTH** (integer) – currently obsolete
- **NDSCREFTPL** (Alpha 1) – Same as for validators: you can define the reference in a separate table (NDFREFS) and enter "`1`" here to show the generator to ignore the fields above and get all needed information from your reference list.
- **NDSCREFTPN** (Alpha 20) – If NDSCREFTPL is set to "`1`", you must state here the reference ID in table NDFREFS.

As this file will grow, it is planned to "normalize" the database design into a flat attributes table to make extensions easier.

(This document is not yet finished. There is still some files to be described)