FIRST CHOICE SOFTWARE

Customization Replicator 1.1.6

Developer Series

Installation and Users Guide

First Choice Software, Inc.
8900 Business Park Drive
Austin, Texas 78759

Web: www.fchoice.com
Phone: (512) 418-2905
Fax: (512) 418-2983

Table of Contents

Overview

The *Developer Series* of products from First Choice Software, Inc. is designed to make it easier to develop and maintain your Clarify system. The **Customization Replicator** (**CR**) is a tool that allows you to easily describe your customizations, and allows you deploy them quickly and without error.

Before You Upgrade to a New Version…

You should be aware of both the version of the product you currently have as well as the version you are upgrading to.

You may upgrade several revision levels at one time. To perform this multiple-upgrade, you must create a list of files to upgrade. Find the section of the manual with instructions for upgrading from your current software revision. Make a list of the required steps. Repeat this process with each update, adding these steps to the list. If a step is required by more than one version, you only have to list it once.

When your list is completed, you should perform all the steps (in the same order you would install the product (schema first, then files and forms, then resource configurations, then code compilation) for a clean install) using the files provided with the most recent version of the product.

For example, suppose you had version 1.1 of a product, and versions 1.1.1, 1.2, and 1.3 have all been released. You now wish to upgrade from version 1.1 directly to version 1.3. Suppose that the versions require the following steps:

Version 1.1.1 requires that you compile files a.cbs and b.cbs.

Version 1.2 requires that you add a new field to the case table, and recompile a.cbs.

Version 1.3 requires that you import a data file d.dat, import a new form file, c.dat (add it to the resource configuration), and compile c.cbs.

To perform the total upgrade in this situation you would perform the following steps:

1. Add the new field to the case table (schema changes always come first)
2. Import the d.dat file (data imports come next)
3. Import the c.dat form, and add it to the proper resource configuration(s).
4. Compile the a.cbs, b.cbs, and c.cbs files.

All of these steps would be accomplished with the files provided with the 1.3 version of the product. If you have questions about this process, please contact First Choice Software.

What's New in Version 1.1.6

This version addresses a small bug that was found in **RCR**.
RCR had a small issue when building the Installer File for global ClearBasic modules. If the global module did not have a subsystem, **RCR** would incorrectly use a default subsystem of *user* in the Installer File.
This issue has been fixed.

To implement this change, simply copy the new *build_cr_file.cbs* file from this package over your old version. It is recommended that you re-run **RCR** to re-create your Installer File.

What's New in Version 1.1.5

This version adds a bug fix to **CR**.

1. **CR** now allows Forms to be loaded from files and will replace existing files if found.

What's New in Version 1.1.4

This version adds a few small enhancements and bug fixes to both **CR** and **RCR**.

2. **CR** and **RCR** now handle Generic Field Ids for view fields. Please refer to the **New View Fields** section later in this document for full details.
3. **RCR** now handles relations on user-defined tables even if the USER_DEFINED keyword wasn't used in the original relation definition.
4. A small issue with the flags field on new view fields using **CR** was discovered. This has been corrected.
5. A bug with **RCR** was corrected that involved having multiple fields on an object that had the same spec_field_id. If an object had multiple fields that had the same spec_field_ids, **RCR** previously would just use the first field. This only happens with objects that contain binary array fields (which aren't even used anymore in Clarify). **RCR** now ignores binary array fields and retrieves the correct field. Under normal circumstances, this issue would never surface. However, depending on how a schema was compiled, and with which tools, it is possible for the schema to exhibit this behavior.
6. A small bug in **CR** was discovered that could cause view definitions to be built improperly. This could happen if a join definition joined a table back to the same table. This has been corrected.
7. Changes have been made in both **CR** and **RCR** to support Clarify 10.x. In previous versions of Clarify, Clarify used two different names for comment fields – "comment" and "comments". As of Clarify 10, they only use one field name, so **CR** and **RCR** were modified to properly handle this Clarify change.

To implement these changes, simply copy the new *build_cr_file.cbs* and *install.*cbs files from this package over your old version. It is recommended that you re-run **RCR** to re-create your installer file.

If you have existing installer files, you should manually edit them to add the placeholder (or definition of) generic field ID for a view field.
For example, if you previous had an entry in your installer file that looked like:

```
[new_view_fields]
;view_name|field_name|from_table|from_field|comment
view_pages|emp_objid|employee|objid|Employee Objid
```

Then you can add an additional placeholder at the end of the line for the generic field id:

```
[new_view_fields]
;view_name|field_name|from_table|from_field|comment|gen_field_id
view_pages|emp_objid|employee|objid|Employee Objid|
```

What's New in Version 1.1.3

This version adds a few small enhancements to **RCR**.

8. RCR now handles subsystem and user labels for global CB code modules. Please refer to the **Compiling Code** section later in this document for full details.
9. These subsystem and user label options are shown in the commented sample usage lines in the generated **RCR** installer file.
10. **RCR** now properly handles user versions of forms that contain spaces. This could be an issue when **CR** compiles CB code modules against forms whose user version contained a space.

To implement this change, simply copy the new *build_cr_file.cbs* file from this package over your old version. It is recommended that you re-run **RCR** to re-create your installer file.

What's New in Version 1.1.2

This version corrects an issue that was discovered in joins for custom views that affected both **CR** and **RCR**.

Basically, it was possible in Clarify for a user to define a join alias in a view that was ***also*** the name of a table within Clarify. While this was (and is in our opinion) a very bad idea, it was allowed in base Clarify. However, **RCR/CR** had issues with it, as **CR/RCR** assumed that the found alias name was in fact the Clarify table, not the join alias. This has now been corrected and **CR/RCR** should allow this in all cases. To see an example of this, the baseline Clarify schema contains a vew named *table_qry_case_view* (view 208) that contains a join whose alias name is *victimcase*. That name (*victimcase*) is ***also*** the name of a base Clarify table!!! Again, we recommend that you not do this, but **CR/RCR** will now support it.

To implement this change, simply copy the new *install.cbs* and *build_cr_file.cbs* files from this package over your old versions. If you have custom views in any of your installer files, you may wish to run **RCR** or **CR** as appropriate.

What's New in Version 1.1.1

This version addresses a couple of small bugs found with the **RCR** program.

1. If you have the same form in multiple resource configurations, the installer file was not built correctly in all cases.
2. If you had double quotes in a form field (such as the form title), **RCR** did not export the dataex-format form file correctly. As such, **CR** would have trouble importing it. As an aside, while this has been addressed, it is against Clarify's standards to do this, and it should be avoided (the double quotes in the form fields) whenever possible.
3. A couple of small documentation bugs were fixed in this manual.
4. A specific warning is now included as to how to avoid the "Out of String" error message that Clarify sometime returns with CBBatch. Please see the "Before you run CR or RCR" section below for more details.
5. Some additional enhancements were made in error reporting for the **CR** program.

To apply the patch, simply copy the new *install.cbs* and *build_cr_file.cbs* files included with this update over your old versions. No other changes are needed.

What's New in Version 1.1

This release of **CR** introduces a number of exciting new enhancements. They are detailed in the appropriate sections of the manual below. A summary of these new features is included in this section.

The new enhancements to **CR** are:

1. You can now specify outer joins (for new joins on views) with the outer half being either side of the join. Previously, only the left side of the join could be outer. This is accomplished by setting the outer_flag to a value of '2'.
2. There are a number of new sections that can be used in an installer file. They are:
   - [add_clarify_list]. This allows you to create a brand new Clarify list (gbst_lst).
   - [add_user_list]. This allows you to create a brand new user-defined list (hgbst_lst).
   - [remove_form_from_rc]. This section allows you to remove a user-defined form from a resource configuration.
   - [remove_form_from_all_rcs]. This section enables you to remove a user-defined form from all resource configurations in which it is located.
   - [remove_form_from_db]. This section removes a user-defined form from all resource configurations, and deletes the form from the database.
   - [add_form_to_rc]. This section enables you to add a user-defined form to a specific resource configuration.

3. The **RCR** program now generates section headers for the new sections, but does not put any data in them (due to the fact that it is impossible to know what they user wants for these sections in a new database). The sections must be filled in manually with a text editor.
4. Better error reporting. In the log file, there are now some better descriptions of warning codes to help an administrator or developer understand what they did incorrect in writing or editing an installer file.
5. If there are any errors in a CB code file compilation, the *cbex.log* file is now returned in the CR log file. This way you can know the compile error immediately. Previously, you had to find and look at the .log file yourself.

To apply this patch, simply copy the two files, *install.cbs* and *build_cr_file.cbs* from this package over the copies installed on your machine. You do not need to edit or alter any of your existing installer files.

What's New in Version 1.0.5

This release fixes a small bug in Reverse CR. If you have a user-defined relation that points from a table to the same table (a site to site relation for example), Reverse CR was not exporting it into the file. This has now been corrected.

To implement this change, copy the build_cr_file.cbs file in this package over your old copy.

What's New in Version 1.0.4

This release fixes a small bug in view field processing. It was possible (for a very small number of views in Clarify), that adding a view field to an existing view could cause the datatype of an existing view field to change. This would only occur in cases where the view field's flags were > 16384. If this happened, you would see an error when using that view.

To implement this change, simply replace the old install.*cbs* with the new one provided.

What's New in Version 1.0.3

This version fixes two tiny bugs in **CR**, and adds a nice new feature. Previously, if you added a user-defined list to the "Clear HGBST List" section, and that list was not defined in the target database, a runtime error would result. Also, the log file displayed the Clarify version, not the user version in the results for form files that were imported. Both of these have been corrected.

Also, the output to the console now shows the names of each data file imported, as well as the form number of the forms, while they are importing. This feedback helps show what work is being done, so a user doesn't wonder "Why is it taking so long? I haven't seen anything in a while." Now you'll see fairly frequent feedback.

To implement this change, simply replace the old install.*cbs* with the new one provided.

What's New in Version 1.0.2

This version adds two enhancements to the **RCR** program, and fixes one small bug.

1) If a user incorrectly marks a field as "Predefined" (Clarify-defined), in a user-defined table, RCR did not previously find that field and export it. Now it does.
2) In the **RCR** program, the list of Clarify tables argument has been made more general. You may specify a relation for the search, or not. If you don't, **RCR** will add it for you.
3) A small bug was found in the dataex files generated for Clarify lists to be exported. This has been fixed.

If you have a previous version of **CR** installed, simply copy the new version of *build_cr_file.cbs*, and regenerate any **CR** files that you wish to have. If your files do not include Clarify lists to be exported, there is no need to run **RCR** again.

What's New in Version 1.0.1

Version 1.0.1 improves the view join processing. Joins on MTM relations can now be processed, and the order of OTM join specification is no longer a specification issue. Also, views created on a mixture of user-defined fields and Clarify-defined fields will be properly sequenced in the view structure.

Both the **CR** program (*install.cbs*) and **Reverse CR** program (*build_cr_file.cbs*) have changed to support these new features.

If you are using version 1.0 of **CR**, you should perform the following steps:

1. You should copy the new versions of *install.cbs* and *build_cr_file.cbs* to your machine.
2. If you have generated any files with Reverse CR, and they exported any custom views, you should regenerate the file with *rcr.bat*.

If you are not a user of version 1.0, you can use the product as described in this manual.

Clarify Customizations

Clarify is an extremely customizable system. This is useful, in that you can change the system to behave exactly the way you need it to. One of these customizations (or a new product from Clarify) can be fairly complex to install on your system, however. And the problem only gets more complicated when you have to install the same set of customizations and schema modifications on many machines.

In a development/production environment, you might have to install the same set of customizations on other developer's machines, test servers, training servers, pre-production servers, and eventually, the production database. Each machine you have to install requires more time, and increases the likelihood that you will make a mistake, costing you more downtime.

Most importantly, when it is time to install the customization on your production machine, you want to make sure to install it as quickly as possible, without any errors.

Installing Your Customizations

Installing customizations in a Clarify system can be fairly complicated. Depending on the customization you are installing, you may have to do one or more of the following:

1. Schema Compilation. You use the DD Editor or DDComp to export your schema, edit it and recompile it. This can be used to add new tables, views, joins, fields, relations, and indexes.
2. Use UI Editor to create a Resource Configuration.
3. Use Data Exchange or UI Editor to import user forms.
4. Use UI Editor to save new user versions of forms.
5. Use UI Editor to put user versions of forms in resource configurations.
6. Use Data Exchange to import data files.
7. Use Policies & Customers to add new user lists, add items to lists, and remove items from lists.
8. Use Policies & Customers to create or modify auto-numbering schemes.
9. Use a bulk loader (such as BCP or SQLLOADER) to load large volumes of data quickly.
10. Use CBEX to compile ClearBasic code.
11. Execute one or more external programs, such as registering components, or compiling a stored procedure.

Each of the steps above can be quite lengthy, and is error-prone. It is easy, for even the most experienced of administrators, to make mistakes when installing a customization. And, as mentioned in the previous section, you have multiple servers, there is no reason to have your people repeating the same steps over and over, and spending what can be a very large amount of their valuable time.

The Customization Replicator

The **Customization Replicator (CR)** is a set of tools that simplifies and greatly speeds up the process described above. There are two primary tools in the **CR**:

1) The **CR** itself. This tool reads a simple ASCII file, the format of which is described in this manual, and applies the changes to the specified database. The tool manipulates the database directly, and does not use DD Editor, DDComp, or UI Editor. It is very important to understand that it performs all of the operations required in **exactly the same manner** as do the Clarify tools. All required adp tables are modified, sequencers and stored procedures are created as needed, and all inserts/updates are performed. **CR** also is aware of the latest Clarify additions. It is able to handle items such as Traveler, ALLOW_NULL, searchable fields (Oracle only), change fields/change dates, and exclusive relations.

   CR performs all necessary data validations. Consider new tables, for example. If you provide the name of a new table, and that name is already used for another table, you will receive a warning. Likewise, you will be warned if you provide an inappropriate table number. If the table you wish to create is already existing in the database, you will be informed of that fact. If the table is not properly formatted in the **CR** file, you will be alerted to that fact. In total, **CR** performs well over 200 validations against your **CR** file before it can be processed in the Clarify database. The **CR** file is also parsed to make sure that the appropriate number of fields are defined for each of the item types included.

2) **Reverse CR (RCR)**. **RCR** can be used to read an existing Clarify database, and *extract* the schema, form, list, and code information, and builds a **CR**-compatible file. Thus, a developer does not need to manually create a **CR** file if they are developing a customization on their machine. They simply use **RCR** to extract the information, edit the **CR** file (usually to delete extraneous schema that was on their machine, but is not needed for the customization), and use that file as input to **CR** to replicate the customization on other machines.

A typical **RCR/CR** scenario is described below.

1) A developer writes and tests a new customization on their Clarify system.
2) They extract the customization using **RCR**. This process creates the following:
   a. A **CR**-format file.
   b. One file for each user-defined form they created, stored in the *forms* directory.
   c. One file for each Clarify or user-defined list they wish to export, stored in the *files* directory.
   d. One file for each global Clarify ClearBasic module they created, and one file for each module file they created, stored in the *cb* directory.
3) The developer edits the created CR file, if needed. They may wish to modify what was extracted, or add to it (for example, to register components).
4) The developer then runs **CR** once for each Clarify database on which they wish to install the extracted customization.

Using The Customization Replicator

The following sections will describe all you need to know to use the **CR/RCR** tools.

Before You Run CR or RCR

Both **CR** and **RCR** are ClearBasic programs. As such, they will use a *clarify.env* file. This can be a standard *clarify.env* file, but you should make sure that it **ALWAYS** contains two very important lines. These lines make sure that you have the maximum stack size possible, and a very large heap. The sample below has 1,000,000 for the heap, but that can be larger if need be. The stack size is the MAXIMUM it can be. Do not try to increase it.

Not setting these settings can cause **CR** or **RCR** to return an “Out of string” error message, due to a limitation in the Clarify *CBBatch* program.

The settings you should include are:

```
CbPublicSize=1000000
CbStackSize=8192
```

Using CR

The Customization Replicator is a versatile tool that will allow you to easily and effortlessly install virtually any customization against a Clarify database. This section will explain a variety of interesting points that you should know about using the **CR**.

1. The **CR** is a ClearBasic program. It does manipulations directly against the Clarify database. It does not use the DD Editor or the UI Editor (although it does use CBEX to compile ClearBasic code, and can use Dataex to import forms and data files).
2. **CR** performs extensive error checking. For example, on schema items, it checks to see if a new item already exists in the database. If so, you will be warned, and the item is bypassed. Also, **CR** checks both the Clarify metadata tables (the ADP tables), and in SQL to make sure that the items are added correctly. If the item is not properly formed, **CR** will inform you of the error, and stop.
3. If **CR** encounters an error, it will report the error, and stop immediately. You can then repair whatever is wrong, and continue with the **CR** installation. **CR** performs a *checkpointing* operation so that it can continue where it left off. If you delete the files in the temp directory (defined later), the checkpoint information will be lost.
4. **CR** uses a file named *install_data.dat* as input. This can be modified in the *install.cbs* file.
5. The results of CR are placed in a log file. The name of the log file is: *install_log.txt*, and the file is located in the temp directory you specify. If you use the Perl installation script, the log file is displayed for you when you finish the installation.

   The log file details each of the changes made (or items bypassed because they already exist in the database). At the end of the log file there will be summary statistics presented. These will tell you the number of items created, the number of files imported, the number of errors and the number of warnings. A sample output log is included near the end of this section.

6. After each run of **CR** you should delete all of the files in the temp directory. It is not required, but strongly suggested.
7. There are two ways to run the **CR** program. The first is to run the CBBatch program manually. The more common way is to run it using the provided Perl script. Both approaches are detailed below.

Manually Running CR

CR is a ClearBasic program. As such, you can execute it with the CBBatch program. The following is the syntax to use to execute the program (note that you should be in the same directory as the *install.cbs* file):

```
<cbbatch_dir>\cbbatch -db_server <server> -db_name <db>
                      -user_name <user> -password <pass>
                      -f install.cbs –r cr_install
                      –as <user> -as <pass> -as <server> -as <db>
                      -as <install_dir> -as <temp_dir>
                      -as <cbbatch_dir> -as <cbex_dir>
                      -as <dataex_dir> -as <input_dat_file>
```

```
Where:
      <cbbatch_dir>      Is the directory containing CBBatch. This is
                         usually the current directory.
      <user>             Is the user name to run the installer with.
                         This is usually “sa”.
      <pass>             Password for the user.
```

```
<server>          Is the name of the database server.
<db>              Is the name of the database.
<install_dir>     Is the current directory.
<temp_dir>        Is the location to write the temporary files
                  and the output log file.
<cbex_dir>        Directory where the CBEX program can be found.
                  This is usually the current directory.
<dataex_dir>      Directory where the Dataex program is found.
                  This is usually the current directory.
<input_dat_file>  This is the install data file from which CR
                  will run
```

Note: If you have a *clarify.env* file in the current directory, you do not need to specify the –db_server, -db_name, -user_name and –password arguments.

Note: If you are running CR often against the same database, you may wish to put the call to cbbatch to run **CR** in a batch file.

Running CR with the Perl Script

Much easier than running CBBatch manually (as in the previous section) is using the provided *cr.bat* file and the *setup.pl* Perl program. These programs ask questions of the user to gather the required information, and then start up the CBBatch program.

To run the Perl script, you must follow the installation instructions found later in this document. If all of the files are properly copied into the directory, the installer can be started by running the *cr.bat* script. For example, type:

```
cr
```

Note that there are no arguments for the program. This will start the *setup.pl* Perl script. This script will ask you several questions. They are detailed here:

```
1. Has mfc42.exe been run on this machine (y,n)?
```

This question is important, because administrators sometimes run the CR on machines that haven't had mfc42.exe run (it is in the client directory). If it hasn't run, Clarify's CBEX program will not run properly. If you **have** run this program on your machine, you can remove the question if you do not wish to see it every time you run the installer.

```
2. What is the system administrator's user name?

3. What is the system administrator's password?

4. What is the server name?

5. What is the database name?
```

These questions are standard Clarify questions. If you are running **CR** on the same database over and over again, you may modify the *setup.pl* script, and put your desired defaults in the variables at the top of the script. If you do, *setup.pl* will present you with the defaults, and you only have to hit Enter to accept them.

```
6. What is the file name of the input file?
```

You should enter the name of the **CR** installer file for the data you wish to install. By default, this is *install_data.dat*. **CR** assumes that the file is located in the current working directory.

```
7. What is the pathname of the temp directory?
```

You should enter the location where the temp files will be run.

After you enter the answers to these simple questions, the **CR** program will be run. First, a test script (*setup_test.cbs*) will be run to insure that the system is set up properly. Then, the main script (*install.cbs*) will be run, using the input file you specified as input to drive the **CR** process. After completion, the log file will be displayed for you to view and analyze.

CR Environment File

The Perl script generates and uses an environment file named *fc.env*. This file contains the answers to the questions from the last time that **CR** was run in the current directory. When the Perl script runs, it reads the file (if present), and offers those values as default answers to the questions. The environment file remembers the last answer to all 7 of the questions asked by the script.

In addition, you may add another line to the ENV file. This line, if present (and set to TRUE), uses the default answers, and automatically runs **CR**, without asking the questions. The name of this variable is *auto_run*, and can be set to TRUE or FALSE.

A sample *fc.env* file ships with the product, and it is defined as follows:

```
auto_run = FALSE
mfc_run = y
login_name = ADMIN
db_password = PASSWORD
db_server = SERVER
db_name = DATABASE
input_file = install_data.dat
temp_dir = c:/tmp
```

Using Reverse CR

Reverse CR (**RCR**) is also a ClearBasic program. It is named *build_cr_file.cbs*, and is located in the **RCR** directory.

The purpose of **RCR** is to extract customizations from a Clarify database, and to build the installer file that you can then use with **CR** to apply the customization to other Clarify databases. In addition to building the installer file, **RCR** will extract data to create external files (data files, form files, ClearBasic code files) as needed.

It is important to understand what **RCR** can do for you. When you run **RCR**, there are certain customization elements that **RCR** will always extract for you. These are:

- User-defined tables (including fields and relations)
- User-defined fields and relations that are added to existing Clarify tables
- User-defined views (including joins and fields)
- User-defined view fields added to existing views

Note: There is a bug in Clarify. Depending on how you add a view field to an existing view (via DD Editor GUI, or via a schema file), Clarify can mark it as either user-defined, or Clarify-defined. **RCR** has an argument (detailed below) that will allow you to always extract view fields from existing Clarify tables.

In most situations, you do not want **RCR** to extract all of the custom forms in a database, and all of the custom code from the database. You only want to extract a specific customization. Usually, the forms and code associated with that customization are in a specific resource configuration.

When you specify a resource configuration to **RCR**, it will extract the following for you:

- All forms associated with that resource configuration
- All user-written ClearBasic code associated with those custom forms
- All global ClearBasic modules

- All users associated with that resource configuration

Finally, there are certain items in your Clarify database, which you might want to apply to another system with **CR**, but that Clarify **never** marks as custom. **RCR** provides a mechanism for you to specify and export those objects, so that they are available to apply against other systems with **CR**. The objects that you can list to export are:

- The forms to put in the save_as list
- Clarify lists
- User-defined lists
- Auto-numbering schemes
- View fields on existing views

There are a few sections in the **CR** installation file that **RCR** simply cannot determine. After you run **RCR**, you can edit the output file, and add them as needed. They include:

- Command-line executions
- Lists to clear (user-defined and Clarify-defined)
- Indexes

RCR is run as follows:

```
<cbbatch_dir>\cbbatch -db_server <server> -db_name <db>
                      -user_name <user> -password <pass>
                      -f build_cr_file.cbs –r build_cr_file
                      –as <out_file> -as <rc_config>
                      -as <save_as_list> -as <list_list>
                      -as <hlist_list> -as <num_schemes>
                      -as <custom_vf_list>

Where:
      <cbbatch_dir>     Is the directory where the cbbatch program is
                        located.
      <user>            Is the user name to run the installer with.
                        This is usually “sa”.
      <pass>            Password for the user.
      <server>          Is the name of the database server.
      <db>              Is the name of the database.
      <out_file>        Is the name of the output file to create.
      <rc_config>       This is the name of the resource configuration
                        to export (forms and code). If this is left
                        blank, no forms or form code will be exported
                        (only custom schema and other non-RC items).
      <save_as_list>    List of forms (comma separated) to not export
                        as form files, but to put into the save_as
                        section
      <list_list>       Names of Clarify lists (comma separated) to
                        export into data files, and include in the
                        generated installer file
      <hlist_list>      Names of user-defined lists (comma separated)
                        to export into data files, and include in the
                        generated installer file
      <num_schemes>     Names of auto-numbering schemes to export into
                        the generated installer file (comma separated)
      <custom_vf_list>  List of views fields to export into the
                        installer file (see below). This argument is
                        sometimes needed because of a small bug in
                        Clarify.
```

Note: If you have a *clarify.env* file in the current directory, you do not need to specify the user, pass, server, and db arguments.

The next few sections discuss some of the extra arguments that you can specify for **RCR**.

Output File

RCR allows you to specify an output file name. If you do not specify a name, **RCR** will use the default name (*install_data.dat*). If a copy of the output file already exists, it will be renamed, with a ".bak" put on the end of the file name.

Resource Configuration

You have the option of specifying a resource configuration, or not. If you only want to export the schema information from your database, do not specify a resource configuration. **RCR** will print a warning, and export the schema-based customizations. If you do specify a resource configuration, **RCR** will find the custom forms in that RC, and will export them, along with any custom ClearBasic code for those forms.

Save As List

When **RCR** runs, it will find all custom forms in the specified resource configuration. But there are times, where you do not need to have the custom form exported.

If you have performed a "save as" on the form, **BUT NOT CHANGED ANY OF THE GUI OF THE FORM**, you do not need to export the form. This is the case when you have custom CB code to compile against the form, but no GUI changes. Clarify requires you to have a custom form. There's really no reason for you to have to create a dataex-format file for the form. Instead, you really just want to add it to the *Save As* section.

For each form number listed in this argument, **RCR** will add it to the *Save As* section. Also, when **RCR** is exporting forms, it will not export the information for this form to a dataex file. This will save time when you run **CR** against other databases, as the *Save As* process is faster than Dataex. You may specify multiple forms, in a comma-separated string. For example,

```
"807, 328"
```

List of Clarify Lists

The *list_list* argument allows you to specify one (or more) Clarify lists to export into dataex-format files, and to add to the *imp_data_files* section. In addition, the list name will be added to the *clear_gbst* section, so that the list is cleared before the new items are inserted. The names of the lists must be comma-separated, and must match the list names as defined by Clarify. For example,

```
"Response Priority Code, Case Type"
```

would export the two lists, and add them to the *imp_data_files* and *clear_gbst* sections.

List of User-Defined Lists

The *hlist_list* argument allows you to specify one (or more) User-defined lists to export into dataex-format files, and to add to the *imp_data_files* section. In addition, the list name will be added to the *clear_hgbst* section, so that the list is cleared before the new items are inserted. The names of the lists must be comma-separated, and must match the list names as defined by the user. For example,

```
"My List1, My List 2"
```

would export the two lists, and add them to the *imp_data_files* and *clear_hgbst* sections.

Numbering Schemes

Auto-numbering schemes are another item that Clarify does not flag as Clarify-defined or user-defined. It is fairly common that you either want to create or update auto-numbering schemes as part of applying your customization.

By placing a list name (or names) in this argument, you cause **RCR** to export data into the proper section of the installer file that records the data for the numbering scheme. For example, if you want to export the current value of the *Case ID* scheme, and the user-defined scheme *Class ID*, you would have the following comma-separated string in the argument:

```
"Case ID, Class ID"
```

View Fields

The **RCR** program automatically locates and extracts view fields when those fields are defined on user-defined (new) views. To have these fields extracted, you have to do nothing.

A small problem arises, however, with view fields added to Clarify-defined views. There is a small bug in Clarify, that causes those view fields to be defined as Clarify-defined (if they are added via a schema file), and as user-defined (if added in the DD Editor GUI).

If you add all your view fields (for existing tables) via the DD Editor GUI, again, you have to do nothing – **RCR** will extract them for you. If, however, you add the view fields to existing views via the schema file mechanism, and you want those fields extracted, you must list them explicitly in this section.

The syntax for this section is to list the view fields as comma-separated strings, with the view name and field name separated by a period. For example,

```
"empl_user.address, empl_user.city"
```

would export two fields from the empl_user view, the address and city fields.

Note: Fields specified in this section can also be Clarify-defined fields. However, they will already be in the database against which the installer file will be run.

CR File Syntax

The **CR** file is a simple ASCII file. It can be edited with your favorite ASCII editor. The next sections describe, in detail, the syntax of a **CR** file.

Note that the separator for items on a given line (with one exception, which is detailed below) is a pipe character ("|"). If a field (on a line) should be empty, make sure that you still have the proper number of pipe characters on a line. For example, in the new table line below, the comment (3rd argument) is blank. Thus, there are two pipes next to each other, with no text specified for the argument.

```
2500|my_table||test_group
```

Comments

Comments are allowed in **CR** files. Any line which starts with a semicolon or a single quote is considered to be a comment, and is ignored by **CR**. When you run **RCR**, comments are placed at the start of the file (header), and at the end (footer). These comments are intended as informational only, and may be deleted, if desired.

RCR Header

The following is a sample header from **RCR**. It contains a copyright statement, the name of the output file, the database server and database name from which the customization information was extracted, the resource configuration from which custom forms were extracted, the user who performed the extraction, and when they extracted the data.

```
; Customization Replicator
; Copyright (C) 2000. First Choice Software, Inc. All Rights Reserved.

; CR-Generated Output File     : install_data.dat
; Generated from database      : ELROND/cl9
; Generated for resource config: test1
; Generated by                 : sa
; Generated at                 : 02/03/2001 05:03:09 PM
```

RCR Footer

The following is a sample footer from **RCR**. It contains the copyright statement, the name of the file, and the date/time that the extraction was completed.

```
; Customization Replicator
; Copyright (C) 2000. First Choice Software, Inc. All Rights Reserved.

; CR-Generated Output File : install_data.dat
; Generation completed at  : 02/03/2001 05:03:10 PM
```

Sections

CR files look very much like *.ini* files. The file is broken up into sections. Each section has a header, which is the name of the section surrounded with square brackets. For example, `[new_table_fields]` is the section header where you will describe fields to add to database tables. The order of the sections is mostly unimportant, but it is recommended that you leave the sections in the order described here (and in the order that is generated by **RCR**). For example, you can change the order of the new fields and new relations sections, but you cannot move the section for compiling ClearBasic code before the section that is responsible for importing form files. Also, it is not required that all sections be listed in any installer file. You only need to list the section headers for the sections you are using. But many people find it easier to include all the section headers, even if some of them have blank sections.

The following is a list of the currently supported sections:

```
[variables]
[new_tables]
[new_table_fields]
[new_table_relations]
[new_indexes]
[new_views]
[new_joins]
[new_view_fields]
[assign_users]
[add_clarify_list]
[add_user_list]
[remove_form_from_rc]
[remove_form_from_all_rcs]
[remove_form_from_db]
[clear_hgbst]
[clear_hgbst]
[add_hgbst]
[clear_gbst]
[add_gbst]
[num_schemes]
[imp_form_files]
[imp_data_files]
[save_user_forms]
[add_form_to_rc]
[compile_cb_code]
```

```
[command_lines]
```

Each of these sections will be described in detail below. At the end of this document, a sample installer file will be provided, with all of the sections containing data.

Variables

This section allows you to describe variables that will be used in the **CR** execution. Currently, there is only one variable defined. That variable is:

```
new_form_version
```

You should always define this variable in a **CR** file. It defines three items for you:

1) This is the name of the resource configuration that **CR** will create for you. If this resource configuration already exists, **CR** will use it for the form/code module manipulation.
2) If you define any users in the *assign_users* section, those users will be added to the specified resource configuration.
3) If you define any forms in the *save_user_forms* section, the new user version for those forms will be the same as the variable defined in this section.

A sample of usage of this variable would be:

```
new_form_version=query1.0
```

New Tables

This section allows you to define new tables in the Clarify system. The tables you create must have a unique table name, and the table number must be in the Clarify-defined range (430-512 or 2000-5000).

For each new table you wish to create, you should add one row to this section. Each row should be entered in the format detailed below:

```
Table number|Table name|Comment|Object group
```

The *table number* is the object number for the new table.

The *table name* is the name of the table, excluding the "table_" that all Clarify tables are prefixed with.

The *comment* is an optional comment which describes the table.

The *object group* is another text field that groups tables together for logical purposes. Any text can be placed in this field. For more information about object groups, please see the Clarify documentation about the data model.

Note: With **CR**, you do not describe the entire table in one place. In this section you list the table header. Then, you list the fields and relations in their own sections. The same process is used for new views.

The following are two valid new table lines for a **CR** file:

```
3600|query_objs|Query objects|FC Query Anything
3601|query_props|Query properties|FC Query Anything
```

New Table Fields

This section allows you to add new fields to either a new table (described in the New Tables section above), or to an existing table in your schema. Just as with schema files and **DDComp**, you do not have to prefix your fields with "x_" as you do when you create fields in the **DD Editor** GUI. The format for a new table field line is as follows:

```
Table number|Field name|Data type|Field length|Field default|Gen Field
ID|Specials|Comments
```

The *table number* is the object number of the table to which you should add the field. The *field name* is the name of the field. It must be unique for the table, and must not be a SQL reserved word.

The *data type* must be one of the following:

1)	char	Variable length strings
2)	char2	Text columns (up to 32K of text). Remember that you may only have one of these per Clarify table
3)	decimal	Currency fields
4)	date	Date/time columns
5)	long	Long integers
6)	float	Floating point numbers

A few other data types are supported, but not commonly used. These include short (Clarify DB type of 1), small (Clarify DB type of 2), binary, and a few other variants of float and char. Please consult the Clarify documentation about the data model for more information.

The *field length* is only used for *char* data fields, and is the maximum length (1-255) of the string. This field is ignored for all other data types.

The *field default* is the default value that CR should place in the field definition. Please see Clarify documentation for valid values for this argument.

The *Gen Field ID* is a special column that you should use only if you know exactly how these work. Clarify uses this column to designate special attributes of the field. For example, objid columns always have a Gen Field ID of 3, and dev columns always have it set to 151. If you are just adding user-defined fields, this column should be left blank.

The specials argument allow you to define some special attributes for the field. As with the Gen Field ID, this feature is for the advanced user. The currently defined special attributes are:

1)	ALLOW_NULL	This field can contain a NULL (always true on Oracle)
2)	SEARCHABLE	Allows for case-insensitive searches (Oracle only, feature is built-in for other databases)
3)	CHANGE_DATE	If any fields marked as CHANGE_FIELD are updated, the current date/time is automatically placed in this field. Only one field per table may be marked as CHANGE_DATE, and it must be a date/time field.
4)	CHANGE_FIELD	If this field is updated, the corresponding CHANGE_DATE field for this table will be updated
5)	CURRENCY	This is a currency field
6)	DONT_DISTR_TO_CLIENT	Don't distribute this field to the client. For Traveler only
7)	DONT_DISTR_TO_SERVER	Don't distribute this field to the server. For Traveler only

One or more special attributes may be defined for a given column, and these should be separated by a comma. The order of the arguments does not matter. For example, a field that is searchable, should not be distributed to the client, and that is a change_field, could have either of the following special arguments:

```
SEARCHABLE,DONT_DISTR_TO_CLIENT,CHANGE_FIELD
CHANGE_FIELD,SEARCHABLE,DONT_DISTR_TO_CLIENT
```

The *comments* are optional comments about the field.

All fields that are added to tables will be added as *User Defined* and *Optional*. In addition, these new fields will be added as the last fields in the table definition (which is required by Clarify).

New Table Relations

This section allows you to add relations between any two tables in a Clarify database. You only need to define the relation once; **CR** will add both the relation and the inverse relation. **CR** supports all relation cardinalities (One to Many, Many to One, One to One (Primary and Foreign), and Many to Many).

For each relation that you wish to add, you must add one line to this section. The format of each line is as follows:

```
Table number|Relation name|Target table number|Inverse relation
name|Cardinality|Exclusive|Comment
```

The *table number* is the object number of the table for the relation.

The *relation name* is the name of the relation. It must be unique for the table, and it is highly recommended (but not required) that it follow the Clarify naming convention of <purpose>2<target_name>. For example, courses_for2contact.

The *target table* number is the object number of the other table in the relation.

The *inverse relation name* is the name of the relation on the target table.

The cardinality describes the type of relation. Remember that the cardinality is for the relation name. The inverse relation will automatically be assigned the inverse cardinality. The valid cardinalities are:

1)	MTO	Many to One
2)	OTM	One to Many
3)	OTOP	One to one (primary side)
4)	OTOF	One to one (foreign side)
5)	MTM	Many to many

The *exclusive* argument is an advanced feature that should only be used if you truly understand Clarify's exclusive relations. Usually, this argument is left blank.

If a relation is an exclusive relation, you must fill in the following pieces of information, which are comma separated:

1)	type_field	This is the name of the long integer field you define to hold the table (object) number
2)	objid_field	This is the name of the long integer field you define to hold the object ID (objid) of each exclusive relation row
3)	exclusive_set	This is the name of the exclusive relation set (Clarify 9.x and later)

Note: When you define exclusive relations, you **MUST** define (in the fields section) the two long integers that are used for the table number and objid fields. If you do not, **CR** will still function properly, but Clarify will not.

Exclusive relations are only defined for Clarify 7.x and later. If you have Clarify 6.x or earlier, you **MUST** leave this argument blank. If you have Clarify 7.x or 8.x, you should only supply the first two of the arguments above (in Clarify 7.x and 8.x there can only be one set of exclusive relations per table. In versions after that you may have multiple sets per table, and **MUST** specify a set name).

For example, if you defined three relations (Clarify 7.x) that use field *focus_type* for the table number and field *focus_objid* for the object ID, the exclusive string would be:

```
FOCUS_TYPE,FOCUS_OBJID
```

If, for version 9.x or later, you placed this in a set called *my_set*, the exclusive string would be:

```
FOCUS_TYPE,FOCUS_OBJID,MY_SET
```

The *comment* is an optional comment describing the relation.

Note: There is currently a requirement for CR that you list the relation so that the table with the **lower object number** be listed first, followed by the table with the higher object number.

A sample of three relations to add to a Clarify database might be:

```
20|user2query_query|3602|query_query2user|OTM||Personal queries for this
user
45|contact2telco_phone|4000|telco_phone2contact|OTM||Phone numbers for
this contact
52|site2telco_phone|4000|telco_phone2site|OTM||Phone numbers for this
customer(site)
```

New Indexes

CR allows you to add indexes to Clarify tables. When you add indexes in this manner, it is just like adding them in the schema file, and compiling with DDComp. Note that some users prefer to add indexes manually in SQL (to better control advanced database features). If you prefer to do that at your site, you should not use this section of the **CR** file.

Each index to be added requires one line be added to this section. The format of an index line is as follows:

```
Table number|Index name|Is unique|Field list
```

Where the *table number* is the object number of the table for which the index should be added.

The *index name* is a unique index name for the index.

The *is unique* field is either the integer 1, which means the index is unique, or 0, which means that it is not.

The *field list* is a comma separated list of field names and relation names (if the relations are MTO or OTOP only!!) that comprise the index. The order of the field/relation names is important; it is the order in which **CR** will construct the index.

For example, the following adds three non-unique indexes to a Clarify database, one of which has more than one field in it:

```
95|active_index|0|x_active
3555|zip_index|0|zipcode
4300|obj_index|0|obj_num,obj_id
```

New Views

This section allows you to add new views to the Clarify database. Each new view to be added will be represented by one line, whose format is as follows:

```
View number|View name|Unique field|Comment|Object group
```

The view number is the object number for the new view. It must fall in the Clarify-defined range for user views (430-511 or 2000-5000).

The *view name* is the name of the view, excluding the “table_” that all Clarify views are prefixed with.

The *unique field* is the name of one of the view fields which makes each view row unique. This field is required.

The *comment* is an optional comment which describes the view.

The *object group* is another text field that groups views together for logical purposes. Any text can be placed in this field. For more information about object groups, please see the Clarify documentation about the data model.

The following is an example of a new view row from a **CR** file:

```
3591|fc_case_scan|objid|Case monitor information|FC SLAM
```

New Joins

This section describes the joins that are required for the new views defined in the previous section. This section has a limitation that it may only be used to describe joins on new views. It may not currently be used to add joins to existing views. For each join to be added to a new view, the following information must be specified:

```
View name|Table 1|Relation 1|Alias1|Table 2|Relation 2|Alias 2|Join
Flag|Comment
```

The *view name* is the name (not the number) of the view that this join will be added to.

Table 1 is the table name (not number) of the first table that participates in the join.

The *relation 1* argument is the name of the relation for *table 1* that describes the join condition.

The *alias1* field is the alias (if one is defined) for *Table 1*

Table 2 is the table name (not number) of the second table that participates in the join.

The *relation 2* argument is the name of the relation for *table 2* that describes the join condition.

The *alias2* field is the alias (if one is defined) for *Table 2*

The *join flag* argument must be either *0* (it is an inner join), *1* (it is a left outer join), 2 (it is a right outer join), or the MTM table name for a MTM join. Note that well over 90% of all joins are inner joins. If this is an outer join, then a value of 1 in this field means that *Table* 1 is the outer table. If it is set to 2, then *Table* 2 is the outer table.
For MTM joins, put the table with the lower type id value as the primary table, and the table with the higher type id value as the secondary table. For example, a join on the mtm_contact2_contract1 table would be defined as follows:

pet_view|contact|caller2contract||contract|contract2contact||mtm_contact2_contract1|*comment*

Note: You can get a listing of the MTM table names (for MTM relations) in one of the first chapters of Clarify's data dictionary guide.

The *comment* argument is an optional argument that describes the join.

Note: For more information about join aliasing, please see the Clarify data modeling documentation.

For example, the following join information describes the six joins used in the First Choice Software product **SLAM**. It describes six joins from the case table to the wipbin, site, user (current owner) tables, and three rows of the gbst_elm table (for priority, severity, and case type). Since the same table is joined to case three times, aliases must be used.

```
fc_case_scan|case|case_wip2wipbin||wipbin|wipbin2case||1|Wipbin of case
fc_case_scan|site|cust_loc2case||case|case_reporter2site||0|Case site
fc_case_scan|gbst_elm|resp_priority2case|gse_priority|case|respprty2gbst_elm||0|Priority
fc_case_scan|gbst_elm|resp_severity2case|gse_severity|case|respsvrty2gbst_elm||0|Severity
fc_case_scan|gbst_elm|case_status2case|gse_status|case|casests2gbst_elm||0|Case status
fc_case_scan|user|curr_owner2case||case|case_owner2user||0|Case owner
```

New View Fields

This section allows you to add fields to views. Unlike the previous section, you may add view fields to both new views, and existing views. Each field to be added to a view must have a row in the **CR** file that looks like:

```
View name|Field name|From table|From field|Comment|Generic Field ID
```

The *view name* is the name (not number) of the view to add the field to.

The *field name* is the name of the field in the view. This may be similar to the name that the field has in the base table (and is helpful to do so), but is not required. The only requirements are that the field name must be unique within the view, and must not be a reserved SQL name.

The *from table* is the name (not number) of the table from which the view field is taken.

The *from field* is the name of the field in the from table.

The *comment* is an optional comment about the view field.

The *Gen Field ID* is a special column that you should use only if you know exactly how these work. If you are just adding user-defined fields, this column should be left blank.

The following are samples of view fields added to the view defined in the previous sections. And one extra view field is added to the end that adds a new view field to an existing Clarify view (site_view) for the notes field from the site table.

```
fc_case_scan|objid|case|objid|Case internal record number|
fc_case_scan|case_id|case|id_number|Case Id number|
fc_case_scan|case_title|case|title|Case Title|
fc_case_scan|case_status|gse_status|title|Case status|
fc_case_scan|case_priority|gse_priority|title|Case priority|
fc_case_scan|case_severity|gse_severity|title|Case severity|
fc_case_scan|case_owner|user|login_name|User login name|
fc_case_scan|creation_time|case|creation_time|Date of case creation|
fc_case_scan|site_id|site|site_id|Site id|
fc_case_scan|wipbin_objid|wipbin|objid|wipbin object ID|
site_view|site_notes|site|notes|Spec consid for the site|
```

Assign Users

This section allows you to add users to the resource configuration that is described in the *Variables* section above. Each user to be added to the RC is given one row, and the login_name of the user is listed as the lone field on the row. For example, to add/move users Marty and Jeanne to the resource configuration, the section would be:

```
[assign_users]
marty
jeanne
```

Add a New Clarify List

This section allows you to add a new Clarify list (to the gbst_lst) table. This is not commonly used, but it is possible that you need to add a new list to the ones already in a database. More common (even for a one-level list) is to add a new user-defined list (see the next section).

If you want to add elements to this newly-created list, you should use the “Add to Clarify Lists” section (detailed below).

To add a new Clarify list, simply supply the name of the list on a line by itself. For example:

```
[add_clarify_list]
new_cl_list
other_cl_list
```

Add a New User-defined List

Using this section you can add a new user-defined list in the database (table_hgbst_lst and table_hgbst_show). Note that currently, **CR** only adds a one-level list, and does not allow you to add sub-lists (lower levels).

If you want to add elements to this newly-created list, you should use the “Add to User-defined Lists” section (detailed below).

Each item that you wish to add to a user-defined list will have a row in the section that has the following format:

```
List_name|Description|Level_name
```

The *list name* is the name of the user-defined list.

The *description* is the optional description of the list.

The *level_name* is the optional name of the 1st level of the list. If this is left blank, the default of “Level 1” is used.

The following is an example of adding some user-defined lists:

```
[add_user_list]
car_colors|This is a list of car colors|
car_options||
car_dealers|A list of possible dealers|First Level
```

Remove a Form From a Resource Configuration

This section allows you to specify a form (user version) and remove it from a particular resource configuraton. It can be a user version of either a Clarify-defined or user-defined form. Each line follows the following format:

Form_id|clarify_version|user_version|resource_config

Where the *form_id* is the form number of the form to remove.

The *clarify_version* is the Clarify version of the form to remove.

The *user_version* is the user version of the form to remove.

The *resource_config* is the name of the resource configuration to remove the form from.

An example section follows:

```
[remove_form_from_rc]
1005|1.0|qa1.0|test_config
1005|1.0|qa1.0|administration
411|9.0|my_version|test_config
```

Remove a Form From all Resource Configurations

There are times that you do not wish to remove a form from just one resource configuration, but from *all* resource configurations. If that is the case, you can use this section to do just that. Each line in this section specifies a user version form. It can be a user version of either a Clarify-defined or user-defined form. Each form so specified will be removed from any and all resource configurations in which it is located.

The following is the format for this section:

Form_id|clarify_version|user_version

Where the *form_id* is the form number of the form to remove.

The *clarify_version* is the Clarify version of the form to remove.

The *user_version* is the user version of the form to remove.

The following is an example of removing two user-defined forms from all resource configurations.

```
[remove_form_from_all_rcs]
420|9.0|my_vers
1901|1.0|qa1.0
```

Remove a Form From the Database

There are occasions where you simply want to remove a user version of a form from a database. It can be a user version of either a Clarify-defined or user-defined form. This includes removing it from all resource configurations **AND** deleting the form so that it cannot be used in the database. To do that, use the format specified for this section:

Form_id|clarify_version|user_version

Where the *form_id* is the form number of the form to remove.

The *clarify_version* is the Clarify version of the form to remove.

The *user_version* is the user version of the form to remove.

The following section is an example of deleting some user-defined forms from the Clarify database.

```
[remove_form_from_db]
1800|1.0|qa1.0
420|9.0|my_vers
```

Clear User-defined Lists

This section allows you to clear out an existing user-defined list in the target database. This can be useful when you have your own values to add to the list, and don't want any of the previous values to be retained. In Clarify, you may always delete items from a user-defined list, as no items are ever related directly to them.

Note: If you simply wish to create a new user-defined list, you should describe that list in a dataex-format file, and import it with the *imp_data_files* section that will be described below. If you need to create such a data file, it can be easily performed with the **RCR** program – see the section on **RCR** below.

To clear a user-defined list, simply include the list name on a line by itself in this section. For example, if we wished to clear two lists named *my_list1* and *my_list2*, the section would be:

```
[clear_hgbst]
my_list1
my_list2
```

Add to User-defined Lists

Using this section, you may add items to user-defined lists. Note that this item can only be used to add items to the first level of user-defined lists. To perform adds on lower levels, format those adds as a dataex file, and import it with the *imp_data_files* section that will be described below. If you need to create such a data file, it can be easily performed with the **RCR** program – see the section on **RCR** below.

Each item that you wish to add to a user-defined list will have a row in the section that has the following format:

```
List name|Element name|Status
```

The *list name* is the name of the user-defined list.

The *element name* is the name of the element to add to the list. This must be a unique name for the first level of the list.

The *status* is the current status of the item in the list. The acceptable values for this argument are:

1)	Active	The list item is active
2)	Inactive	The list item is inactive
3)	Default	This is the default item for the list. If you make a new item the default, the previous default will be changed to Active

The following example adds two items to the list *my_list1*, and one new item to the list *my_list2*:

```
my_list1|High|Active
my_list1|Medium|Default
my_list2|Green|Default
```

Clear Clarify Lists

This section can be used to clear out an existing Clarify list. Since items in Clarify lists (stored in the gbst_lst and gbst_elm tables) can still be related to rows in the database, you may not always simply delete the rows from the Clarify list. If the item to be cleared is related to at least one row in the database, it is not deleted, but is made *Inactive* (which leaves it in the database for viewing existing items, but does not allow it to be used for new selections). If it is not related to any database rows it is deleted. This procedure is followed for each row in the Clarify lists specified.

For each list to be cleared, the following format must be specified:

```
List title|Relation name
```

The *list title* is the name of the Clarify list to clear.

The *relation name* is the name of the relation (found on the gbst_elm table) that the list uses to relate to the other Clarify table. For example, if you wished to clear the Case Type list, you would specify the following:

```
Case Type|call_type2case
```

The reason for using this relation is the *call_type2case* relation is the relation that the gbst_elm table uses to locate the case rows that use a specific case type.

The following table lists most of the Clarify lists, and the relations they use:

List Name	Relation Name
Response Priority Code	resp_priority2case
Problem Severity Level	resp_severity2case
Case Type	call_type2case
Resolution Code	resolution2close_case
Contact Role	contact_role2contact_role
CR Type	bug_type2bug
CR Priority	priority2bug
CR Domain	domain2bug
CR Class	class2bug
CR Severity	severity2bug
Task Priority	priority2task
Task Type	type_task2task
Dialogue Priority	dialogue_pty2dialogue
Communication Response	resp2communication

Add to Clarify Lists

Using this section, you may add items to Clarify lists. Each item that you wish to add to a Clarify list will have a row in the section that has the following format:

```
List name|Element name|Status
```

The *list name* is the name of the user-defined list.

The *element name* is the name of the element to add to the list. This must be a unique name for the list.

The *status* is the current status of the item in the list. The acceptable values for this argument are:

1)	Active	The list item is active
2)	Inactive	The list item is inactive
3)	Default	This is the default item for the list. If you make a new item the default, the previous default will be changed to Active

The following example adds two new priorities and a new case type:

```
Response Priority Code|2 - Medium|Default
Response Priority Code|1 - Low|Active
Case Type|Customer Whining|Active
```

Add or Modify Numbering Schemes

This section allows you to add a new auto-numbering scheme to the target database, or update an existing scheme. For each scheme you wish to add or update, you must add a line to the section with the following format:

```
Scheme name|Format|Start number|End number|Current number
```

The *scheme name* is the name of the auto-numbering scheme. If it already exists in the target database, then an update is performed. Otherwise, an insert is done.

The *format* is the format of the scheme. Please see the Clarify System Administrator's Guide for details on acceptable formats. For a simple counter, use *%i* as the format.

The *start number* is the first number in the sequence. Usually this number is set to 1.

The *end number* is the last number of the sequence.

The *current number* is the next number to use (for the counter). If this is to be a new sequence, it is usually set to the same number as the start number. Be aware, though, that if the target database is already using this sequence, resetting the sequence number can cause uniqueness problems.

As an example, add a new sequence for Modules (that is just a simple counter), and change the format of the Case ID scheme.

```
Module Number|%i|1|200000|1
Case ID|Case %i|1|200000|1
```

Note: This section does not currently allow you to set any advanced features of numbering schemes, such as padding or month names. If you wish to change these, you may format/export a new dataex object, and import it with the *imp_data_files* section.

Import Form Files

This section allows you to import forms (that were previously exported) from dataex-format files into a Clarify database. In addition, it will add the user version of the form to the resource configuration that is specified with the *new_form_version* variable. The form files that you import with this section are the files created when you *export* forms with Clarify's UI Editor, or when you use the **RCR** tool from this package.

Each form you wish to import should be saved in a separate file, and should be represented with one line in this section of the **CR** file. Do not include multiple forms in one file. This requirement is stipulated for two reasons:

1) If there is any issue or error with importation, you know which form and form file caused the problem. This makes problem resolution much easier.
2) You can only add one user version of a form to a resource configuration with each line in the **CR** file. Having multiple forms in one file would mean that not all of the forms would be properly added to the resource configuration.

The syntax for importing files is as follows:

```
File name|Form number|Clarify version|User version
```

The *file name* is the path and name of the form file. Quite often you will define subdirectories of the directory from which you run **CR** to hold the form files. In this case, you will define the path to be a relative path to the current location. For example *forms\qa2211.dat*
The *form number* is the object number of the form to import. This should match the form number that is found in the *id* field of the *window_db* object in the form file. For example, if you were importing a new user version of the *New Case* form, the form number would be 411.

The *clarify version* is the version number that Clarify has assigned to the form for this version of Clarify. For new user versions of existing Clarify forms, it can be found either with UI Editor, or in the *ver_clarify* field of the *window_db* object in the form file. For new (fully custom) forms, the Clarify version is usually set to 1.0.

The *user version* is an optional argument. If you specify it, it should be the user version of the form, as found in the *ver_customer* field of the *window_db* object in the form file. If you do not specify a user version for any line in this section, it is given a user version that matches the *new_form_version* variable that is defined in the *variables* section.

The following are two sample rows of form files to import. The first one has a user version supplied. The second one does not (and uses the value in the variable at the top of the file):

```
qa\forms\1947.dat|1947|1.0|fc1.0
forms\cc11650.dat|11650|7.0
```

Import Data Files

This section allows you to specify data files to import. Most of the time, these files will be in Clarify's dataex format. In addition, for very large files, you can list ASCII files (in **BCP** format for Sybase and SQL Server, or **SQLLOADER** format for Oracle) to perform bulk imports.

For simple dataex imports, you will specify one line in this section for each file. The format of the line is:

```
File name
```

The *file name* is the full path and name of the file to import. As with form files, you may use absolute paths (including drives on PC operating systems), or relative paths. Relative paths are relative to the directory in which you started **CR**.

If you have bulk data loads to perform, you may specify a file to import with the following syntax:

```
Table|File name
```

The *table* is the name of the Clarify table in which the imported data is to be placed. The file's format must be a pipe delimited ("|") file that matches the table schema exactly, including objid columns, searchable columns (for Oracle only), and relations. Care must be exercised with this option to make sure that the data will successfully import.

The *file name* is the full path and name of the file to import.

It is often desirable to use this method, when possible. A **BCP** load of 74,000 rows into an empty table might take a minute, where the same data would take 15 minutes or more to import with dataex. The limitation of this approach is that only inserts are possible, and it is complicated to generate the data file.

The following are two rows of a sample data file that would import zipcodes into a new zipcode table which is defined as the objid, zipcode, address line 1, state, country, and time zone.

```
268435457|92274|100 Palms|CA|USA|PST
268435458|92276|1000 Palms|CA|USA|PST
```

A sample of some data files to import in this section follows. Note that the first line would import the data via a bulk loader, and the other lines would use dataex.

```
zipcode|import\zipcode.imp
import\commcenter_config.dat
import\commcenter_lists.dat
```

Save New User Versions of Forms

This section allows you to save new user versions of forms, without having to import a form file. This option is used when the customization has code (usually in ClearBasic) that you wish to apply to a form, but do not have any GUI changes to make to the form. Since ClearBasic requires that you have a custom version of the form against which to compile code, you have to save a new user version of the form.

It is permissible to save a user version of a form (with no changes), export the form, and import it with the `imp_form_files` section, but it is not necessary. By specifying the form number in this section, the **CR** program will create a copy of the baseline form, give it a user version, and save it in the resource configuration specified in the *Variables* section. The format of the section is:

```
Form number
```

The *form number* is the form to save in the database. **CR** will determine the Clarify version for you automatically, and the user version will be set to the value in the *new_form_version* variable that is defined in the *variables* section.

A sample of data for this section might look like this:

```
321
328
334
340
404
```

Note: The forms specified in this section **MUST** be form numbers predefined by Clarify. You may not use this section for user-defined forms.

Adding a Form to a Resource Configuration

There are times, that you need to add a form to a resource configuration. Remember that if you either import a form file or save a new user version (using the sections above), **CR** will automatically add them to the resource configuration specified in the variable section for you.

This section is used for two primary purposes:

1) A form that you added with one of those two sections must be added to a *different* resource configuration, or
2) You have a dataex-style file with multiple forms in it. In this case, you can use the "Import Data File" section to import the forms into the database. This will create the forms, but will not add them to any resource configurations.

```
Each line in this section will add one user version of a form (Clarify-
defined form or user-defined form) to a specified resource
configuration. The lines look as follows:
```

Form_id|clarify_version|user_version|resource_config

Where the *form_id* is the form number of the form to remove.

The *clarify_version* is the Clarify version of the form to remove.

The *user_version* is the user version of the form to remove.

The *resource_config* is the name of the resource configuration to add the form to.

The following is an example of adding forms to resource configurations:

```
[add_form_to_rc]
1005|1.0|qa1.0|test_config
1005|1.0|qa1.0|administration
411|9.0|my_version|test_config
```

Compiling Code

This section allows you to compile ClearBasic code against user-defined forms, as well as global code modules. Each line in this section represents one code module to be compiled. Each global code module to be compiled should be in the following format:

```
file_name G clarify subsystem –N global_label
```

If not using a label, the syntax should be:

```
file_name G clarify subsystem
```

Note: Unlike the other sections in this file, the lines of the code compilation section use a space as a separator, not a pipe.

The *file_name* is the path and name of the global module to compile. The path can be either absolute (including path on PC platforms), or relative (to the current directory).

For global modules, the letter *G* is then used (for global), followed by the keyword *clarify*.

Finally, the *global_label* is a unique string that uniquely identifies this global module. The "-N" string must be included if using a global label.

For example, the following two lines describe two global modules to compile.

```
cb\cc_global.cbs G clarify cc_global
cb\cc_phone_globals.cbs G clarify appsrv –N phone_globals
```

For each form module to compile, you should include a line such as the following:

file_name F form_num clarify_version user_version

The *file_name* is the path and name of the form module.

For form modules, the letter *F* is then used.

The *form_num* is the form number of the form to compile the module against.

clarify_version is the Clarify version of the form. If you do not know the version, just put in 1.0. **RC** will automatically figure out the baseline's Clarify version, and use it.

The *user_version* is the user version of the form to compile the code against. If the *user_version* contains spaces, then it should be enclosed inside double quotes.

For example, the following two lines would compile to form modules:

```
cb\cc412.cbs F 412 5.0 cc3.1
cb\cc423.cbs F 423 5.0 “cc 3.1”
```

Execute Command-line Programs

This section of the file allows you to execute command-line programs necessary to complete your customizations. Common uses for this section include registering components, and compiling stored procedures and triggers. Each line of this section contains a command-line to execute.

It is important to understand that Clarify’s ClearBasic *shell* command is asynchronous. As such, it is not possible to get status back from a command-line execution. Each program will be started, and the **CR** program cannot wait until it completes, or know the status of the execution. If you have more than one program to execute, and they are dependent on each other, they should be placed in one batch or shell script file (to handle the sequential execution).

The following is a sample of a component being registered, followed by the compilation of a SQL Server stored procedure.

```
[command_lines]
regsvr32 c:\tmp\ietimer.ocx
isql –Usa –Psa –SMY_SERVER < sproc.sql
```

A Sample CR File

This section includes a sample **CR** installer file. It is intended for illustration purposes only, and does not include all of the sections that you can use with **CR**. This particular file (and the log in the next section) are the files that are used at First Choice Software to install our product **Query Anything**.

```
[variables]
new_form_version=query1.0

[new_tables]
;table_num|table_name|comment|obj_group
3500|fc_string|Locale-based strings for First Choice Software Customizations|FC
I18N
3501|fc_list_hdr|Header record for First Choice Software list object|FC I18N
3502|fc_list_level|One level of a First Choice Software list|FC I18N
3503|fc_list_elm|One element in a First Choice Software list|FC I18N
3504|fc_list_locelm|One locale string for an element in a First Choice Software
list|FC I18N
3505|fc_locale|Locale Details|FC I18N
3600|query_objs|Query objects|FC Query Anything
3601|query_props|Query properties|FC Query Anything
3602|query_query|One specific query|FC Query Anything
3603|query_clause|A clause for a specific query|FC Query Anything
3604|query_param|A parameter for a clause for a specific user|FC Query Anything

[new_table_fields]
;table|field_name|type|array_size|fld_default|gen_fld_id|others|comment
;new fc_string fields
3500|id|long||null|||String ID
3500|string|char|255|null|||Text of the string
3500|locale|char|20|null|||Indicates the locale of the string; e.g., EN_US=US
English, JA_JP=Japanese in Japan
3500|application|char|80|null|||Application where string is used
3500|dev|long||null|||Row version number for mobile distribution purposes

;new fc_list_hdr fields
3501|title|char|80|null|||Name of the list
3501|description|char|255|null|||Description of the list
3501|application|char|80|null|||What is the list for?
3501|dev|long||null|||Row version number for mobile distribution purposes

;new fc_list_elm fields
3503|rank|long||null|||The position of this element in the level
3503|state|long||null|||The state of the element. 0 = Default. 1 = Active. 2 =
Inactive
3503|fc_use|char|20|null|||Only used by the i18n list gui. Do not depend on this
field having a value.
3503|fc_use2|char|20|null|||Only used by the i18n list gui. Do not depend on
this field having a value.
3503|dev|long||null|||Row version number for mobile distribution purposes

;new fc_list_locelm fields
3504|title|char|20|null|||String for the locale element
3504|locale|char|20|null|||Locale for the locale element
3504|other|char|80|null|||Other string for the locale element
3504|dev|long||null|||Row version number for mobile distribution purposes

;new fc_locale fields
3505|locale_name|char|20|null|||3505 field
3505|abday|char|255|null|||3505 field
3505|day|char|255|null|||3505 field
3505|abmon|char|255|null|||3505 field
3505|mon|char|255|null|||3505 field
3505|d_t_fmt|char|255|null|||3505 field
3505|d_fmt|char|255|null|||3505 field
3505|t_fmt|char|255|null|||3505 field
3505|am_pm|char|255|null|||3505 field
3505|t_fmt_ampm|char|255|null|||3505 field
3505|upper|char|255|null|||3505 field
3505|lower|char|255|null|||3505 field
3505|space|char|255|null|||3505 field
```

```
3505|cntrl|char|255|null|||3505 field
3505|punct|char|255|null|||3505 field
3505|digit|char|255|null|||3505 field
3505|xdigit|char|255|null|||3505 field
3505|blank|char|255|null|||3505 field
3505|toupper|char|255|null|||3505 field
3505|tolower|char|255|null|||3505 field
3505|yesexpr|char|255|null|||3505 field
3505|noexpr|char|255|null|||3505 field
3505|int_curr_symbol|char|20|null|||3505 field
3505|currency_symbol|char|10|null|||3505 field
3505|mon_decimal_point|char|10|null|||3505 field
3505|mon_thousands_sep|char|10|null|||3505 field
3505|mon_grouping|long||null|||3505 field
3505|positive_sign|char|10|null|||3505 field
3505|negative_sign|char|10|null|||3505 field
3505|int_frac_digits|long||null|||3505 field
3505|frac_digits|long||null|||3505 field
3505|p_cs_precedes|long||null|||3505 field
3505|p_sep_by_space|long||null|||3505 field
3505|n_cs_precedes|long||null|||3505 field
3505|n_sep_by_space|long||null|||3505 field
3505|p_sign_posn|long||null|||3505 field
3505|n_sign_posn|long||null|||3505 field
3505|decimal_point|char|20|null|||3505 field
3505|thousands_sep|char|20|null|||3505 field
3505|grouping|long||null|||3505 field
3505|dev|long||null|||Row version number for mobile distribution purposes

;new query_objs fields
3600|query_object|char|64|null|||Object to do query on
3600|out1|char|255|null|||Output (display) field 1
3600|out2|char|255|null|||Output (display) field 2
3600|out3|char|255|null|||Output (display) field 3
3600|out4|char|255|null|||Output (display) field 4
3600|out5|char|255|null|||Output (display) field 5
3600|out6|char|255|null|||Output (display) field 6
3600|out7|char|255|null|||Output (display) field 7
3600|out8|char|255|null|||Output (display) field 8
3600|out9|char|255|null|||Output (display) field 9
3600|out10|char|255|null|||Output (display) field 10
3600|out11|char|255|null|||Output (display) field 11
3600|out12|char|255|null|||Output (display) field 12
3600|path1|char|255|null|||Output (path) field 1
3600|path2|char|255|null|||Output (path) field 2
3600|path3|char|255|null|||Output (path) field 3
3600|path4|char|255|null|||Output (path) field 4
3600|path5|char|255|null|||Output (path) field 5
3600|path6|char|255|null|||Output (path) field 6
3600|path7|char|255|null|||Output (path) field 7
3600|path8|char|255|null|||Output (path) field 8
3600|path9|char|255|null|||Output (path) field 9
3600|path10|char|255|null|||Output (path) field 10
3600|path11|char|255|null|||Output (path) field 11
3600|path12|char|255|null|||Output (path) field 12
3600|sort1|char|255|null|||Sort (display) field 1
3600|sort2|char|255|null|||Sort (display) field 2
3600|sort3|char|255|null|||Sort (display) field 3
3600|sort4|char|255|null|||Sort (display) field 4
3600|sort5|char|255|null|||Sort (display) field 5
3600|sort6|char|255|null|||Sort (display) field 6
3600|spath1|char|255|null|||Sort (path) field 1
3600|spath2|char|255|null|||Sort (path) field 2
3600|spath3|char|255|null|||Sort (path) field 3
3600|spath4|char|255|null|||Sort (path) field 4
3600|spath5|char|255|null|||Sort (path) field 5
3600|spath6|char|255|null|||Sort (path) field 6
3600|alias_name|char|64|null|||Alias name to display for table. Example:
Solution instead of probdesc.
3600|dev|long||null|||Row version number for mobile distribution purposes

;new query_props fields
3601|prop_name|char|20|null|||Query property name
```

```
3601|prop_path|char|255|null|||Query property path
3601|prop_list|char|30|null|||List to use for the property
3601|list_type|long||null|||Should we use application list (=0), or user-defined
list (=1)
3601|field_type|long||null|||What data type is the property? string = 0, integer
= 1, float = 2, date = 3, decimal = 4
3601|dev|long||null|||Row version number for mobile distribution purposes

;new query_query fields
3602|query_object|char|64|null|||Object to do query on
3602|query_name|char|255|null|||Name of the query
3602|shared|long||null|||Is the query shared (=1), or personal (=0)
3602|sort1|char|255|null|||Sort field 1 for the query
3602|sort2|char|255|null|||Sort field 1 for the query
3602|sql_stmt|char2|16|null|||SQL Statement for the query
3602|asc_desc|long||null|||Is the query ordered ascending (0) or descending (1)?
3602|dev|long||null|||Row version number for mobile distribution purposes

;new query_clause fields
3603|operation|char|30|null|||Clause operation
3603|value|char|255|null|||Clause value
3603|parameterized|long||null|||Is the query clause parameterized (=1), or not
(=0)
3603|property|char|20|null|||Clause property name
3603|dev|long||null|||Row version number for mobile distribution purposes

;new query_param fields
3604|the_user|char|30|null|||User for this value
3604|value|char|255|null|||Parameter value
3604|dev|long||null|||Row version number for mobile distribution purposes

[new_table_relations]
;table_num|rel_name|target_table_num|inv_rel_name|cardinality|exclusive|comment
20|user2query_query|3602|query_query2user|OTM||Personal queries for this user
3502|level2fc_list_elm|3503|elm2fc_list_level|OTM||Elements for this list level
3502|level2fc_list_hdr|3501|hdr2fc_list_level|OTOF||Relates top level to header
3502|child2fc_list_elm|3503|parent2fc_list_level|OTOF||The element that this
level was called from
3503|locs2fc_list_locelm|3504|locelm2fc_list_elm|OTM||The locale strings for
this element
3600|query_objs2query_props|3601|query_props2query_objs|OTM||Query object
properties
3602|query_query2query_clause|3603|query_clause2query_query|OTM||Query clauses
for this query
3603|query_clause2query_param|3604|query_param2query_clause|OTM||The parameters
for this clausee

[new_indexes]
;table_num|index_name|is_unique|field_list

[new_views]
;view_num|view_name|unique_field|comment|obj_group

[new_joins]
;view_name|table1|relation1|alias1|table2|relation2|alias2|join flag|comment

[new_view_fields]
;view_name|field_name|from_table|from_field|comment|gen_field_id

[assign_users]
sa

[add_clarify_list]
; list_name


[add_user_list]
; list_name|description|level_name


[remove_form_from_rc]
; form_id|clarify_version|user_version|resource_config
```

```
[remove_form_from_all_rcs]
; form_id|clarify_version|user_version


[remove_form_from_db]
; form_id|clarify_version|user_version
[clear_hgbst]
;list_name

[add_hgbst]
;list_name|elm_name|status

[clear_gbst]
;list_title|rel_name

[add_gbst]
;list_title|elm_title|status

[imp_form_files]
; Query Anything
qa\forms\1947.dat|1947|1.0|fc1.0
qa\forms\at2200.dat|2200|1.0|at1.0
qa\forms\qa2210.dat|2210|1.0|query1.0
qa\forms\qa2211.dat|2211|1.0|query1.0
qa\forms\qa2212.dat|2212|1.0|query1.0
qa\forms\qa2213.dat|2213|1.0|query1.0
qa\forms\qa2214.dat|2214|1.0|query1.0
qa\forms\qa2215.dat|2215|1.0|query1.0
qa\forms\qa2216.dat|2216|1.0|query1.0
qa\forms\qa2217.dat|2217|1.0|query1.0
qa\forms\qa2218.dat|2218|1.0|query1.0
qa\forms\qa2219.dat|2219|1.0|query1.0

[imp_data_files]
qa\files\msg_box_strings.dat
qa\files\path_edit_strings.dat
qa\files\query_config.dat
qa\files\query_lists.dat
qa\files\query_strings.dat

[save_user_forms]
328
807

[add_form_to_rc]
; form_id|clarify_version|user_version|resource_config

[compile_cb_code]
; file_name G clarify subsystem
; file_name G clarify subsystem -N label
; file_name F form_id clarify_version user_version

qa\cb\query_global.cbs G clarify query_global
qa\cb\query_global_fd.cbs G clarify query_global_fd
qa\cb\at_2200_fd.cbs G clarify at_2200_fd
qa\cb\lists.cbs G clarify i18n_lists
qa\cb\string.cbs G clarify i18n_string

qa\cb\1947.cbs F 1947 1.0 fc1.0
qa\cb\at2200.cbs F 2200 1.0 at1.0
qa\cb\qa2210.cbs F 2210 1.0 query1.0
qa\cb\qa2211.cbs F 2211 1.0 query1.0
qa\cb\qa2212.cbs F 2212 1.0 query1.0
qa\cb\qa2213.cbs F 2213 1.0 query1.0
qa\cb\qa2214.cbs F 2214 1.0 query1.0
qa\cb\qa2215.cbs F 2215 1.0 query1.0
qa\cb\qa2216.cbs F 2216 1.0 query1.0
qa\cb\qa2217.cbs F 2217 1.0 query1.0
qa\cb\qa2218.cbs F 2218 1.0 query1.0
qa\cb\qa2219.cbs F 2219 1.0 query1.0
qa\cb\qa328.cbs F 328 5.0 query1.0
qa\cb\qa807.cbs F 807 5.0 query1.0
```

A Sample CR Output Log

```
Begin installation processing at: 02/19/2001 11:53:56 AM

[02/19/2001 11:53:56 AM] Start variable validation
  The new resource config query1.0 has been added.
  Variable 'new_form_version' validated.

[02/19/2001 11:53:57 AM] Start new table processing
  Add table_fc_string
  Adding field name objid
  Add table_fc_list_hdr
  Adding field name objid
  Add table_fc_list_level
  Adding field name objid
  Add table_fc_list_elm
  Adding field name objid
  Add table_fc_list_locelm
  Adding field name objid
  Add table_fc_locale
  Adding field name objid
  Add table_query_objs
  Adding field name objid
  Add table_query_props
  Adding field name objid
  Add table_query_query
  Adding field name objid
  Add table_query_clause
  Adding field name objid
  Add table_query_param
  Adding field name objid

[02/19/2001 11:53:58 AM] Start new table field processing
  Adding field name id
  Adding field name string
  Adding field name locale
  Adding field name application
  Adding field name dev
  Adding field name title
  Adding field name description
  Adding field name application
  Adding field name dev
  Adding field name rank
  Adding field name state
  Adding field name fc_use
  Adding field name fc_use2
  Adding field name dev
  Adding field name title
  Adding field name locale
  Adding field name other
  Adding field name dev
  Adding field name locale_name
  Adding field name abday
  Adding field name day
  Adding field name abmon
  Adding field name mon
  Adding field name d_t_fmt
  Adding field name d_fmt
  Adding field name t_fmt
  Adding field name am_pm
  Adding field name t_fmt_ampm
  Adding field name upper
  Adding field name lower
  Adding field name space
  Adding field name cntrl
  Adding field name punct
  Adding field name digit
  Adding field name xdigit
  Adding field name blank
  Adding field name toupper
  Adding field name tolower
  Adding field name yesexpr
  Adding field name noexpr
  Adding field name int_curr_symbol
  Adding field name currency_symbol
  Adding field name mon_decimal_point
  Adding field name mon_thousands_sep
  Adding field name mon_grouping
  Adding field name positive_sign
  Adding field name negative_sign
  Adding field name int_frac_digits
  Adding field name frac_digits
  Adding field name p_cs_precedes
  Adding field name p_sep_by_space
  Adding field name n_cs_precedes
  Adding field name n_sep_by_space
  Adding field name p_sign_posn
  Adding field name n_sign_posn
  Adding field name decimal_point
  Adding field name thousands_sep
  Adding field name grouping
  Adding field name dev
  Adding field name query_object
  Adding field name out1
  Adding field name out2
  Adding field name out3
  Adding field name out4
  Adding field name out5
  Adding field name out6
  Adding field name out7
  Adding field name out8
  Adding field name out9
  Adding field name out10
  Adding field name out11
  Adding field name out12
  Adding field name path1
  Adding field name path2
  Adding field name path3
  Adding field name path4
  Adding field name path5
  Adding field name path6
  Adding field name path7
  Adding field name path8
  Adding field name path9
  Adding field name path10
  Adding field name path11
  Adding field name path12
  Adding field name sort1
  Adding field name sort2
  Adding field name sort3
  Adding field name sort4
  Adding field name sort5
  Adding field name sort6
  Adding field name spath1
  Adding field name spath2
  Adding field name spath3
  Adding field name spath4
  Adding field name spath5
  Adding field name spath6
  Adding field name alias_name
  Adding field name dev
  Adding field name prop_name
  Adding field name prop_path
  Adding field name prop_list
  Adding field name list_type
  Adding field name field_type
  Adding field name dev
  Adding field name query_object
  Adding field name query_name
  Adding field name shared
  Adding field name sort1
  Adding field name sort2
  Adding field name sql_stmt
  Adding field name asc_desc
```

```
  Adding field name dev
  Adding field name operation
  Adding field name value
  Adding field name parameterized
  Adding field name property
  Adding field name dev
  Adding field name the_user
  Adding field name value
  Adding field name dev

[02/19/2001 11:53:59 AM] Start new table relation processing
  Adding relation name query_query2user
  Adding relation name user2query_query
  Adding relation name elm2fc_list_level
  Adding relation name level2fc_list_elm
  Adding relation name level2fc_list_hdr
  Adding relation name hdr2fc_list_level
  Adding relation name child2fc_list_elm
  Adding relation name parent2fc_list_level
  Adding relation name locelm2fc_list_elm
  Adding relation name locs2fc_list_locelm
  Adding relation name query_props2query_objs
  Adding relation name query_objs2query_props
  Adding relation name query_clause2query_query
  Adding relation name query_query2query_clause
  Adding relation name query_param2query_clause
  Adding relation name query_clause2query_param

[02/19/2001 11:54:00 AM] Start new table index processing

[02/19/2001 11:54:00 AM] Start new view processing

[02/19/2001 11:54:00 AM] Start new view join processing

[02/19/2001 11:54:00 AM] Start new view field processing

[02/19/2001 11:54:00 AM] Clear hgbst lists

[02/19/2001 11:54:00 AM] Add new hgbst lists

[02/19/2001 11:54:00 AM] Clear gbst lists

[02/19/2001 11:54:00 AM] Add new gbst lists

[02/19/2001 11:54:03 AM] Assign users to new resource config
 Assigning sa to query1.0

[02/19/2001 11:54:03 AM] Start form file import
 Form '1947' saved as user version '1.0'. Successful.
 Form '2200' saved as user version '1.0'. Successful.
 Form '2210' saved as user version '1.0'. Successful.
 Form '2211' saved as user version '1.0'. Successful.
 Form '2212' saved as user version '1.0'. Successful.
 Form '2213' saved as user version '1.0'. Successful.
 Form '2214' saved as user version '1.0'. Successful.
 Form '2215' saved as user version '1.0'. Successful.
 Form '2216' saved as user version '1.0'. Successful.
 Form '2217' saved as user version '1.0'. Successful.
 Form '2218' saved as user version '1.0'. Successful.
 Form '2219' saved as user version '1.0'. Successful.

[02/19/2001 11:56:28 AM] Start data file import
 File 'msg_box_strings' import. Successful.
 File 'path_edit_strings' import. Successful.
 File 'query_config' import. Successful.
 File 'query_lists' import. Successful.
 File 'query_strings' import. Successful.

[02/19/2001 11:57:29 AM] Save new user version of forms
 Form '328' saved as user version 'query1.0'. Successful.
 Form '807' saved as user version 'query1.0'. Successful.

[02/19/2001 11:57:30 AM] Compile cb code modules
```

```
 Module 'query_global' compiled. Successful.
 Module 'query_global_fd' compiled. Successful.
 Module 'at_2200_fd' compiled. Successful.
 Module 'lists' compiled. Successful.
 Module 'string' compiled. Successful.
 Module '1947' compiled. Successful.
 Module 'at2200' compiled. Successful.
 Module 'qa2210' compiled. Successful.
 Module 'qa2211' compiled. Successful.
 Module 'qa2212' compiled. Successful.
 Module 'qa2213' compiled. Successful.
 Module 'qa2214' compiled. Successful.
 Module 'qa2215' compiled. Successful.
 Module 'qa2216' compiled. Successful.
 Module 'qa2217' compiled. Successful.
 Module 'qa2218' compiled. Successful.
 Module 'qa2219' compiled. Successful.
 Module 'qa328' compiled. Successful.
 Module 'qa807' compiled. Successful.

Processing complete at: 02/19/2001 11:59:07 AM
 Added 302 objects.
 Imported 36 files.
 0 objects were not added.
 0 warnings were reported.
```

Implementation

This section provides detailed requirements, what files are included in this product, installation and other implementation considerations.

Requirements

This version of **CR** requires the following:

Clarify Version:	7.0 or later
Clarify Tools:	Data Exchange (*dataex*)
	ClearBasic Batch (*cbbatch*)
	ClearBasic Exchange (*cbex*)
Other Tools:	*BCP* from SQL Server/Sybase (if performing bulk loads)
	more.exe, pwd.exe, perl.exe from MKS toolkit (if using *cr.bat* script)

Packaging

Customization Replicator is shipped to you as a zip file.

Installation Tree

It is recommended that you uncompress the zip file containing **CR** into an fchoice subdirectory created at the top of the Clarify install tree. For example on NT, should your Clarify server install tree be “c:\clarify\fchoice”, then:

1. Unzip into “c:\clarify\fchoice\cr” directory.

After uncompressing, you will have the following installation tree:
c:\clarify\fchoice\cr

The following files are provided with this product:

File Name	Purpose
cr_user.pdf	This document
CR	Directory for the CR program
RCR	Directory for the RCR program

sample	Samples directory

The following files are provided in the *CR* directory:

File Name	Purpose
cr.bat	Batch file to run installer
setup.pl	Perl script for running the installer
setup_test.cbs	Sample ClearBasic program used by perl script to make sure that CBBatch is properly configured
install.cbs	CR program
fc.env	Sample environment file for automating *setup.pl*

The following files are provided in the *RCR* directory:

File Name	Purpose
rcr.bat	Batch file to run RCR program
build_cr_file.cbs	RCR program

The following files are provided in the *samples* directory:

File Name	Purpose
sample_data.dat	Sample CR installer file

Manual Installation

Note: It is highly recommended that you first install **CR** on a test system and get familiar with it before installing it on a production system.

The **CR** files should be installed on any machine that can execute the Clarify data exchange tool (*dataex*), the ClearBasic Batch Program (*cbbatch*), and the Clarify CleaBasic Compiler (*cbex*). You will also need to run the MKS toolkit programs listed above, but only if you want to use the more automated Perl Script provided. If you want to execute the **CR** and **RCR** programs manually as CBBatch programs, you do not need the MKS toolkit installed.

Once you have unzipped the **CR** package, there are a few steps that you should follow.

- Copy the following files into the CR subdirectory (the directory that contains the *install.cbs* file). Remember, that the DLL files and the Clarify EXE files are dependent on your operating system and database system. If you need to apply the same changes to both a SQL Server and Oracle database (uncommon, but it can happen), you'd have to have two directories set up, with the **CR** programs placed in both of them.
    - All of the DLL files from the Clarify client directory (**.dll*).
    - The *cbbatch.exe* program (from the Clarify server/rulemgr directory).
    - The *cbex.exe* program (from wherever you installed the CB compiler).
    - The *dataex.exe* program (from the clarify server/dbadmin directory).
    - The *perl.exe*, *more.exe*, and *pwd.exe* programs (from the MKS directory). This is only needed if you'll use the Perl script provided.
- You should edit the *rcr.bat* files. In it you should change the path to the *cbbatch* program to reflect the proper path on your machine.
- If you are using the Perl installer script, you should edit the *fc.env* file. In it (near the top), you will see several variables. If you want the Perl installer to offer you defaults each time you run it, you

may edit these variables, and put in acceptable values. If you don't want to set the defaults, you do not need to edit this file (remember that the Perl program will remember the values for you the first time you run it). The variables are:

```
auto_run = FALSE
mfc_run = y
login_name = ADMIN
db_password = PASSWORD
db_server = SERVER
db_name = DATABASE
input_file = install_data.dat
temp_dir = c:/tmp
```

Limitations

There is only one known issue with **CR**. You must run **CR** on a PC machine. It can certainly install customizations, or extract customizations from any Clarify databases (even on UNIX servers), but the executables should be run on a PC.

Performance

There are no known performance issues with **CR.**

Internationalization

There are no known I18N issues with **CR.**

How to Contact Us

For more information about other First Choice Software, Inc. products, or if you have any questions about the **CR** product, please contact us at:

First Choice Software, Inc.
8900 Business Park Drive
Austin TX 78759-7404
(512) 418-2905
support@fchoice.com
www.fchoice.com