

BARRODALE COMPUTING SERVICES LTD.

The Universal File Interface (UFI)

User's Guide

Version 1.0.0.4, October 7, 2011

The Universal File Interface (UFI): User's Guide

© Barrodale Computing Services Ltd.

<http://www.barrodale.com>

Table of Contents

Documentation Conventions.....	i
Typographical Conventions	i
Icon Conventions	ii
What's New in This Version?	iii
Introduction.....	1
A Simple Example	2
List the Locators.....	3
Create the Template Table	3
Create the Virtual Table	4
Define the Locator-Column Mapping	4
Specify the File(s)	4
Validate the Relationship	5
Do Some Queries	5
Installation Instructions	6
Install the UFI DataBlade	6
Setting up the License Key	8
Using the Universal File Interface Demo Programs	9
The SQL API.....	10
UFI_add_column.....	10
UFI_add_file.....	10
UFI_clear_ordered	11
UFI_create_index.....	11
UFI_drop_index	12
UFI_list_tables	12
UFI_list_files	12
UFI_list_locators	12
UFI_make_managed.....	14
UFI_mark_ordered.....	14
UFI_register_adapter	15
UFI_validate.....	15
UFI_validate_file	16
Taking Advantage of Ordered Data	17
Indexing UFI Tables.....	19
Server-side Indexes	19
Adapter-side Indexes	19
Non-Adapter-Specific Locators	21

“alias” locator	21
“ufi_index” locator	21
UFI Adapters.....	22
Adapter Frameworks	23
CSV Adapter.....	24
Locator Format.....	24
Other Notes.....	24
DBF Adapter	25
Locator Format.....	25
GDAL Adapter	26
The GDAL Data Model.....	26
Locators supported by the GDAL Adapter	26
File constants:	26
Dimensional indexes	29
Geographic Coordinates	29
Band Values.....	30
Supporting Additional Raster Formats	31
HDF5 Adapter.....	32
HDF5 Terminology (from HDF User’s Guide)	32
Locator Format.....	32
Group Matching in a Locator.....	34
NetCDF Adapter	37
NetCDF Terminology (from the Unidata site)	37
Locator Format.....	37
Troubleshooting.....	41
Glossary	42
Appendix 1: Performing Table Joins With UFI Tables	43
Appendix 2: The UFI Table Schema.....	45

Documentation Conventions

This section defines the conventions used in this document. The conventions include typographical conventions and icon conventions.




Typographical Conventions

This manual uses the following typographical conventions:

Convention	Meaning
KEYWORD	Programming language keywords (i.e., SQL, C keywords) appear in a serif font.
<i>italics</i>	<i>New terms, emphasized words, and variable values appear in italics.</i>
<code>italics</code>	
User input	Computer generated text (e.g., error messages) and user input appear in a non-proportional font.

Icon Conventions

This manual uses the following icon conventions to highlight passages in the manual¹:

Icon	Label	Description
	Warning:	Identifies paragraphs that contain vital instructions, cautions, or critical information.
	Important:	Identifies paragraphs that contain significant information about the feature or operation that is being described.
	Tip:	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described.

¹ This manual follows the icon conventions used in IBM Informix manuals.

What's New in This Version?

The following table lists the features that have been added to this version of the the Universal File Interface (UFI).

Feature	Manual Sections Where Feature is Described.
This is the first version of this manual.	

Introduction

Many “technologists” (e.g., scientists, engineers, and other researchers) avoid using databases to manage their data; a principal reason given is that they often have very large data files which are too cumbersome/difficult/slow/costly to load into a database. These files come in a variety of formats: [HDF5](#), [NetCDF](#), [NITFS](#), [FITS](#), etc. At BCS we have developed a solution to this problem²: the Universal File Interface (UFI), a database extension based on the IBM Informix Virtual Table Interface (VTI).

VTI is a technology that supports making external datasets appear as tables to SQL queries and statements³. UFI is a BCS database extension for querying the contents of external data files as though they were rows in a database table. UFI makes a file look like a set of database tables, so “UFI-managed tables” are actually *virtual* database tables. Consequently, users of UFI can perform SQL SELECT queries on their files without having to first load them into a database!

UFI consists of the following components:

1. A set of file-type-independent DataBlade routines and data types, forming the core product,
2. A set of file-type-dependent external programs, called adapters, each one responsible for handling a different type of file, and
3. A set of system tables for keeping track of UFI virtual tables, columns, and adapters.

Each of the DataBlade routines is described in the SQL API section and the various adapters are described in subsequent sections. The system tables are described in the Appendix.

We illustrate how UFI can be used to query a file using a simple example defined in the next section.

² We are indebted to Mike Folk (President of the HDF Group, <http://hdfgroup.org>) and Gerd Heber (IT Manager of the Oxford-Man Institute, <http://www.oxford-man.ox.ac.uk/index.html>) for several helpful technical discussions during the formative phase of this project.

³ It addresses the same needs as the *SQL/MED*, or Management of External Data, extension to the SQL standard as defined by ISO/IEC 9075-9:2003. SQL/Med is already implemented for DB2.

A Simple Example

To see how UFI can be used to query a file, consider the following simple CSV (comma separated value) file, which we will assume is stored on the file system in a file called `/home/miscFiles/employees.csv`.

```
EmployeeNumber,Name,Department,AnnualSalary
1,Bob,HR,80000.0
2,Carol,Executive,90000.0
3,Ted,Accounting,85000.0
4,Alice,IT,75000.0
```

We will use UFI to query this file as if the data were actually stored in the following database table:

```
> CREATE TABLE employees(
employee_number INTEGER,
employee_name VARCHAR(100),
department varchar(20),
annual_salary FLOAT);
```

and as if the following query were performed:

```
> SELECT * FROM employees WHERE
    annual_salary > 80000.0;
```

employee_number	employee_name	department	annual_salary
2	Carol	Executive	90000.0
3	Ted	Accounting	85000.0

With UFI we build a virtual database table based on the CSV file, and then perform queries on this virtual database table in the same way we would as if it were a real database table. A number of steps are involved:

1. Determine which “parts” of the CSV file can be mapped to table columns. With UFI, these “parts” are referred to by file [“locators”](#). In the case of CSV files the relationship is very simple: the “parts” are simply the individual strings that appear between commas, and the locators used to identify the parts are strings of the form “`coli`”, where “*i*” is the position number, starting at 0, of the respective part.
2. Create a (real) “template” table “`employees_template`” that has the same apparent structure as the virtual table we want. This template table is really nothing more than a table definition. We will not be loading data into it; we will just be using it as a model, or template.

Note that if your database did have a real table with the same structure as the virtual table then you could skip this step.

3. Specify that we want UFI to create a virtual table “employees” based on the template table “employees_template” and that this virtual table will be backed by a CSV (as opposed to a netCDF, HDF, FITS, etc.) file.
4. Specify the mapping between file locators and table columns.
5. Specify the names of each of the files we want to back the “employees” table. In this case there is just one file, but in general there could be any number.
6. Validate the relationship. This is an optional step to check that the files actually exist and that the locators actually exist in the file.
7. Query away!

Here are the steps again, this time in more detail:

List the Locators

```
EXECUTE FUNCTION ufi_list_locators(  
    '/home/miscFiles/employees.csv', 'csv');  
  
(expression) ROW('lvarchar', 'col0', 'EmployeeNumber')  
(expression) ROW('lvarchar', 'col1', 'Name')  
(expression) ROW('lvarchar', 'col2', 'Department')  
(expression) ROW('lvarchar', 'col3', 'AnnualSalary')  
  
4 row(s) retrieved.
```

The output of the `ufi_list_locators` function provides a list of the datatypes, locators(`col0`, `col1`, ...) and column names⁴ of the locators in the `'/home/miscFiles/employees.csv'` file. The `'csv'` argument tells UFI to use the adapter program registered to handle CSV files.

Create the Template Table

```
CREATE TABLE employees_template(  
    employee_number INTEGER,  
    employee_name VARCHAR(100),
```

⁴ The csv adapter assumes that the first line of the CSV file contains column names.

THE UNIVERSAL FILE INTERFACE (UFI) USER'S GUIDE

```
    department VARCHAR(20),  
    annual_salary FLOAT);  
Table created.
```

We have chosen to define a table structure that includes all four of the locators from the CSV file. Note that in general we don't need to include in our table all of the locators that are in the file. Furthermore, the order of the columns in the table, and the names we give them, do not have to correspond to the locator names or the order in which `ufi_list_locators` lists the locators.

Create the Virtual Table

```
execute procedure  
ufi_make_managed('employees','csv','employees_template');  
Routine executed.
```

With the `ufi_make_managed` routine we're simply telling UFI to create a virtual table called "employees", backed by one or more CSV files, using the structure of the real table "employees_template" as a model.

Define the Locator-Column Mapping

```
execute procedure  
ufi_add_column('employees', 'employee_number', 'col0');  
Routine executed.
```

```
execute procedure  
ufi_add_column('employees', 'employee_name', 'col1');  
Routine executed.
```

```
execute procedure  
ufi_add_column('employees', 'department', 'col2');  
Routine executed.
```

```
execute procedure  
ufi_add_column('employees', 'annual_salary', 'col3');  
Routine executed.
```

We use the `ufi_add_column` function to map the locators we're interested in (in this case, all of them) to the various columns in the "employees" table.

Specify the File(s)

```
EXECUTE PROCEDURE ufi_add_file('employees', 'csv1',  
    '/home/miscFiles/employees.csv');  
Routine executed.
```

We use the `ufi_add_file` function to indicate which files are backing the virtual table. In this case there is just one file, but in general we could associate more than one file with the table. The second parameter to `ufi_add_file` (in

this case, "csv1") provides a file alias that we can "store" in one of the virtual table columns to indicate which file a particular row came from.

Validate the Relationship

```
EXECUTE PROCEDURE ufi_validate('employees');  
Routine executed.
```

We use the `ufi_validate` procedure to check that the file(s) associated with the virtual table exist and are accessible, and that the locators associated with the virtual table columns are all legitimate.

Do Some Queries

```
SELECT * FROM employees;
```

employee_number	employee_name	department	annual_salary
1	Bob	HR	80000.00000000
2	Carol	Executive	90000.00000000
3	Ted	Accounting	85000.00000000
4	Alice	IT	75000.00000000

4 row(s) retrieved.

```
SELECT * FROM employees WHERE annual_salary > 80000.0;
```

employee_number	employee_name	department	annual_salary
2	Carol	Executive	90000.00000000
3	Ted	Accounting	85000.00000000

2 row(s) retrieved.



In the preceding example we used a CSV file to illustrate how simple it is to build a UFI table. However, almost every DBMS has very straightforward mechanisms for loading CSV files into tables, so in reality we would have probably just loaded the CSV file into a real table rather than use UFI. On the other hand, many, more complex, file types (netCDF, HDF5, FITS, GRIB, etc.) cannot so easily be loaded into a real database table, but the steps in UFI for mapping such files to virtual tables are just as simple as in our example above.

Installation Instructions

The following sections assume that you are already running an IBM Informix database instance and a 32 bit x 86 Linux platform. If this is not the case, you have some options:

- 1) You can download the free-for-production-use version of Informix for Linux, the “**Informix Innovator-C Edition for Linux 32**”, from https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=swg-ixnce&S_CMP=rnav. (Note: registration is required, but free.)
- 2) You can download the free-for-nonproduction-use developer version of Informix for Linux, the “**Informix Developer Edition 11.5 for Linux x86 32**”, from https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=ifxids&S_CMP=rnav. (Note: registration is required, but free.)
- 3) If you don't want to install Informix you can instead try the “**Informix / Universal File Interface (UFI) VMware Appliance**”, as discussed in http://www.barrodale.com/bcs/downloads/ufi_ids_appliance/UFI_InstallVMware.pdf.

If you're running Informix on an architecture other than 32 bit x86 Linux and would like to try UFI, please contact BCS at BCSInfo@barrodale.com for more options.



If you have chosen option 3 above, then complete installation and testing instructions are provided in the [Informix / Universal File Interface \(UFI\) VMware Appliance](#) document. If you have chosen either option 1 or option 2, then please continue reading this chapter.

Install the UFI DataBlade

The UFI DataBlade is packaged as a zip file, `UFI_INSTALL.zip`; a trial version can be requested by sending an email to BCSInfo@barrodale.com. This zip file should be copied to `$INFORMIXDIR/extend`, where `$INFORMIXDIR` has been set to the Informix software home directory.

The following instructions must be executed as user “`informix`”. It is assumed that UFI is to be registered into an existing database, which we will refer to as “`dbname`”.

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

1. cd into directory \$INFORMIXDIR/extend

```
$ cd $INFORMIXDIR/extend
```

2. Unzip the file.

```
$ unzip UFI_INSTALL.zip
```

3. If the DataBlade is to be installed by another user, in particular a user who does not have DBSA privileges, then you must either:

- a. Modify \$INFORMIXDIR/etc/\$ONCONFIG to set the parameter IFX_EXTEND_ROLE to 0. (Note, \$ONCONFIG is set to the name of the onconfig file for your Informix instance).

1. or

- b. Grant the “extend” role to the Informix user who will be registering the UFI DataBlade into a database.

```
> GRANT extend TO SomeInformixUser;
```

4. As user informix, build an external dbspace⁵ for use by UFI⁶:

```
% cd $INFORMIXDIR/extend/UniversalFileInterface.1.0.0.0  
% ./buildExtSpace  
External space successfully created.  
%
```

5. As user informix, edit the file \$INFORMIXDIR/etc/\$ONCONFIG and add the line:

```
VPCLASS ufi,noyield,num=1
```

6. Register the UFI DataBlade into the appropriate database (*dbname* in the following example) using `blademgr` (user input is shown in **reverse video** for clarity):

```
% blademgr  
SERVERNAME>list dbname  
There are no modules registered in database dbname.  
SERVERNAME>show modules
```

⁵ Nothing is actually stored in the external dbspace. It is simply required as part of the syntax used by Informix.

⁶ The buildExtSpace script uses `tsh`, so if that shell is not installed on your system you will need to either modify the script or execute the few lines in the script one by one.

THE UNIVERSAL FILE INTERFACE (UFI) USER'S GUIDE

```
9 DataBlade modules installed on server bcslinuxdev2:
  UniversalFileInterface.1.0.0.0   wfs.1.00.UC1
                                bts.2.00         binaryudt.1.0
                                ifxrltree.2.00      mqblade.2.0
                                ifxbuiltins.1.1      LLD.1.20.UC2
                                Node.2.0 c
```

A 'c' indicates DataBlade module has client files.

If a module does not show up, check the prepare log.

```
SERVERNAME>register UniversalFileInterface.1.0.0.0 dbname
```

```
Register module UniversalFileInterface.1.0.0.0 into
database dbname? [Y/n]Y
```

Registering DataBlade module... (may take a while).

DataBlade UniversalFileInterface.1.0.0.0 was successfully registered in database dbname.

```
SERVERNAME> <ctrl c>
```

␣

Setting up the License Key

Unless you are using an evaluation version of the UFI DataBlade, a license key is needed for each computer on which the IBM Informix server runs⁷. The license key is provided to the UFI DataBlade by adding the following lines to the `.bashrc` file in the `informix` account⁸:

```
export UFI_LICENSE_KEY
UFI_LICENSE_KEY=license_key_value
```

If there is a line in the file that reads

```
# User specific aliases and functions
```

place the license key statements right below that line.

Next, restart the IBM Informix server. A simple way to do this is to re-log into the `informix` account⁹ and execute the following commands (assuming that the `informix` account uses the bash shell):

```
oninit > oninit.out 2>&1 &
```

⁷ Evaluation versions are time-limited and do not require a license key.

⁸ If the `informix` account runs with some shell other than bash, then the means for setting the `DBXTEN_LICENSE_KEY` environment variable will be different. For example, if `csh` or `tcsh` is used, then “`setenv DBXTEN_LICENSE_KEY license_key_value`” will need to be placed in the `.cshrc` file in the `informix` account.

⁹ If you're already logged in as `informix` you will need to log out and log in again so that the new license-related environment variables get set.

The license key is dependent on your machine's hostname and IP address. If either of these change, you will need to contact [Barrodale Computing Services Ltd.](#) for a new license key.

Using the Universal File Interface Demo Programs

A package of netCDF and HDF5 files, including some test scripts, can be downloaded from the BCS Web site at http://www.barrodale.com/bcs/downloads/ufi_ids_appliance/UFIDemo.tar.gz.

To try UFI using these files, perform the following steps. It is assumed that you are executing these steps as a user who has connect and resource permission on the “dbname” database, and that UFI has been registered into that database.

1. Set the environment variable UFITEST to a directory into which you wish to download the test file package, then cd to that directory.

```
% cd $UFITEST
```

2. Unpack the test file package that you downloaded from our website.

```
% tar xvzf UFIDemo.tar.gz
```

3. cd into the netcdf directory and run the SSTDemo script. This script will introduce you to the main features of UFI.

```
% cd netCDF  
% ./SSTDemo
```

The SQL API

This section defines each of the functions and procedures that are part of the UFI DataBlade.



Note that, as a general rule of thumb, in specifying table and column names in these procedures it is safer to use lower-case identifiers rather than mixed case or upper case names, as Informix by default converts these identifiers to lower case.

UFI_add_column

```
procedure UFI_add_column(lvarchar UFITable,  
lvarchar columnName,  
lvarchar locator);
```

This procedure is used to add or replace *table-file* mapping information for a particular column of a UFI table.

UFITable	The name of the UFI table to which a column is to be added. The table must have been created through an earlier call to UFI make managed .
columnName	The name of the column whose information is to be set or replaced. The column must exist in the template table that was specified in the earlier call to UFI make managed .
locator	The locator of the object in the file to be associated with the UFI table column. Generally speaking, locator formats are adapter-specific – e.g., the locators used with an HDF5 file will look different from those that are used with a CSV or NetCDF file. However, some locators – notably alias and ufi_index are common to all adapter types.

UFI_add_file

```
procedure UFI_add_file(lvarchar UFITable,  
lvarchar alias,  
lvarchar fileName);
```

This procedure is used to associate [dataset](#) information with a UFI table that was previously created through [UFI make managed](#). The procedure is called once for each file in the dataset.

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

UFITable	The name of the UFI table to which dataset information is to be associated. The table must have been created through an earlier call to UFI make managed .
alias	A user-provided key (alias) for the file that may be used as a column value in the UFI-managed table through the use of the alias locator. Each <code>fileName</code> associated with the same <code>UFITable</code> must have a different <code>alias</code> .
fileName	The absolute path name of one of the files comprising the dataset to be associated with the UFI table.

UFI_clear_ordered

```
procedure UFI_clear_ordered(lvarchar UFITable,  
lvarchar columnNameList);
```

This procedure removes the status information created by an earlier call to [UFI mark ordered](#). See the section [Taking Advantage of Ordered Data](#) (page 17) for more information.

UFITable	The name of the UFI table for which column ordering status information is to be removed.
columnNameList	A comma-separated list of UFI table column names that have been previously marked as ordered by a call to UFI mark ordered .

UFI_create_index

```
procedure UFI_create_index(lvarchar UFITable,  
lvarchar columnNameList);
```

This procedure is used to create an adapter-side index. See the section [Indexing UFI Tables](#) (page 19) for more information on using indexes with UFI tables.

UFITable	The name of the UFI table for which an adapter side index is to be created. The table must have been created through an earlier call to UFI make managed .
columnNameList	A comma-separated list of one or more UFI table column names that are to be indexed. This comma-separated list is

	used to identify the index, as described in the section Indexing UFI Tables (page 19). The columns must have been created through earlier calls to UFI_add_column .
--	---

UFI_drop_index

```
procedure UFI_drop_index(lvarchar UFITable,
lvarchar columnNameList);
```

This procedure is used to drop an adapter-side index. See the section [Indexing UFI Tables](#) (page 19) for more information on using indexes with UFI tables.

UFITable	The name of the UFI table for which an adapter side index is to be dropped. The table must have been created through an earlier call to UFI_make_managed .
columnNameList	A comma-separated list of one or more UFI table column names that comprise the index to be dropped. The columns must have been created through earlier calls to UFI_add_column .

UFI_list_tables

```
function UFI_list_tables();
```

This function returns a list of the UFI tables currently defined in the database.

UFI_list_files

```
function UFI_list_files(lvarchar UFITable);
```

This function returns a list of the files currently associated with a particular UFI table.

UFITable	The name of the UFI table for which the name(s) of the underlying files are sought. The table must have been created through an earlier call to UFI_make_managed .
----------	--

UFI_list_locators

```
function UFI_list_locators(lvarchar fileName,
lvarchar interpretation);
```

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

This function returns a list of the locators currently available for reference in a specific file, given a specific [interpretation](#) to be used in determining the locators.

fileName	The name of the file for which the locators are being sought.
interpretation	A string used to identify the interpretation used in deriving the locators for the file. This interpretation must have been previously associated with the database through a call to UFI_register_adapter ¹⁰ .

The locators are listed, one per line, using the following format:

ROW (dataType, locatorName, locatorDescription)

where the three fields are strings defined as follows:

dataType

The datatype of the locator. Currently supported UFI datatypes (and their corresponding Informix datatypes) are:

UFI Datatype	Informix Datatype
boolean	integer
int	integer, smallint
int8	bigint, int8
double	double precision, float, smallfloat
string	lvarchar
timestamp	datetime year to fraction(5)
interval_ds	interval day to fraction(5)
interval_ym	interval year to month

The Informix datatype of any column that is associated with a particular locator (though [UFI_make_managed](#) and [UFI_add_column](#)) should match (according to the table above) the

¹⁰ UFI_register_adapter does not need to be called for the hdf5 or netCDF interpretations, as these adapters are registered automatically by blademgr.

datatype returned by
UFI_list_locators.

locatorName

The name of the locator. This value is used in calls to [UFI_add_column](#) when defining the mapping between a column of a UFI table and a locator.

locatorDescription

A description of the locator. These descriptions are adapter-specific.

UFI_make_managed

```
procedure UFI_make_managed(lvarchar UFITable,
lvarchar interpretation,
lvarchar UFI_TemplateTable);
```

This procedure builds a UFI-managed table.

UFITable	The name of the UFI table to create.
interpretation	A string used to identify the particular interpretation. This interpretation must have been previously associated with the database through a call to UFI_register_adapter .
UFI_TemplateTable	The name of a real (non-UFI) table with the same structure (columns and column types) as the UFI table being created. This template table may possibly be empty or it may contain real data. Possibly it may exist only for the purposes of defining the structure for one or more UFI tables.

UFI_mark_ordered

```
procedure UFI_mark_ordered(lvarchar UFITable,
lvarchar columnNameList,
lvarchar direction,
lvarchar method);
```

This procedure creates UFI table status information to indicate that a certain set of columns are actually stored in sorted order in the underlying file. See the section [Taking Advantage of Ordered Data](#) (page 17) for more information.

UFITable	The name of the UFI table for which column ordering status information is to be created. The table must have
----------	--

	been created through an earlier call to UFI make managed .
columnNameList	A comma-separated list of UFI table column names whose values together form an ordered list. The columns must have been created through earlier calls to UFI add column .
direction	The value 'ascending' or 'descending', reflecting the ordering of the data in the file.
method	Currently, 'binary' is the only value supported.

UFI_register_adapter

```
procedure UFI_register_adapter(lvarchar pathname,  
lvarchar interpretation);
```

Register an adapter to handle a particular [interpretation](#) of a type of data file. This registration needs to be done just once per database per interpretation. For the HDF5 and NetCDF adapters the registration is done automatically as part of installation (using `blademgr`).



pathname	The absolute pathname, on the database server, of the (binary) adapter program responsible for handling a particular type of file in a particular way. The pathname must start with <code>\$INFORMIXDIR</code> or <code>\$UFI_ADAPTERS</code> . If you specify <code>\$UFI_ADAPTERS</code> in a path, then you must ensure that this environment variable is set in the environment of the process that starts Informix; you may need to change the Informix startup script (e.g., <code>/etc/init.d/informix</code>).
interpretation	A string used to identify the particular interpretation. This value is used in other API procedures to identify the specific interpretation.

UFI_validate

```
procedure UFI_validate(lvarchar UFItable);
```

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

This procedure checks that the columns in the specified UFI table are consistent with the datasets that have been associated with the table through previous calls to [UFI add file](#).

UFITable	The name of the UFI table to be validated. The table must have been created through an earlier call to UFI make managed , with column-locator mappings defined by calls to UFI add column and one or more files associated with it through calls to UFI add file .
----------	--

UFI_validate_file

```
procedure UFI_validate_file(lvarchar UFITable,  
lvarchar fileName);
```

This procedure is similar to [UFI validate](#), except that it doesn't require that [UFI add file](#) be called first.

UFITable	The name of the UFI table to be validated. The table must have been created through an earlier call to UFI make managed , with column-locator mappings defined by calls to UFI add column .
fileName	The name of the file to be validated against.

Taking Advantage of Ordered Data

If a column (or set of columns) is marked as being ordered using the [ufi_mark_ordered](#) procedure, adapters that are based on the [Grid Framework](#) can perform more advanced searches along a particular dimension to optimize the processing of `WHERE` clauses in the absence of indexes. The signature of the [ufi_mark_ordered](#) procedure, described [earlier](#), is the following:

```
procedure UFI_mark_ordered(lvarchar UFITable,  
lvarchar columnNameList,  
lvarchar direction,  
lvarchar method);
```

The procedure has the following restrictions:

- 1) The *columnNameList* argument can be a single column, or a comma separated list of columns.
- 2) All the columns referenced in a particular call to [ufi_mark_ordered](#) must depend on the same, single dimension.
- 3) The columns are listed in most major to least major order.
- 4) If there are multiple columns, they must be strictly ordered. However, you can have multiple calls to [ufi_mark_ordered](#) that pertain to the same dimension and the adapter will use the one that has the most dimensions matched in the `WHERE` clause.
- 5) If you violate restrictions 4 or 5, you will not get an error message; your query will, without warning, produce incorrect results.
- 6) The *direction* parameter should be 'ascending' or 'descending', but 'increasing' and 'decreasing' are accepted as well.
- 7) The *method* can be coded as 'binary', 'quadratic', and 'cubic', but only `binary` is currently supported. Support for quadratic and cubic searches will be added at a later date.

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

Example

Consider the following example of an ordered data file:

A	B	C	D	E
1	-1	0	3	5
1	2	1	2	2
1	3	-3	2	0
2	0	1	1	3
2	2	-1	1	2
3	5	0	1	-1

It would be valid to mark either “A”, or “A,B” as “increasing” columns, and “D”, or “D,E” as “decreasing” columns, but none of B, C, or E could be marked as “increasing” or “decreasing” by themselves.

The [ufi clear ordered](#) procedure removes the information stored by an earlier [ufi mark ordered](#) call.

Indexing UFI Tables

There are two types of indexing that can be used with UFI tables: “server-side” indexing and “adapter-side” indexing.

Server-side Indexes

Server-side indexes are indexes (b-tree or r-tree) implemented by Informix. You create a server-side index on a UFI table identically to how you create an index on a regular table. There are some caveats to server-side indexes though:

- Server-side indexes are limited to 32 bit values (roughly 4 billion rows).
- Server-side indexes must only be used with UFI tables that represent a single file.
- Querying using server-side indexes can be very slow, returning approximately 50 rows a second¹¹.

On the other hand, a server-side index can be used to answer questions like

```
SELECT count(*) FROM satellitemapix WHERE red = 3 AND blue =4;
```

with great speed and efficiency because the server can produce the answer by examining just the data in the index¹², without using UFI at all.

You will likely need to use query directives to force the optimizer to use indexes on UFI tables. You should also be aware that the indexes will not automatically be updated if the underlying data file is changed.

Adapter-side Indexes

Adapter-side indexes are indexes implemented as files, completely handled by UFI adapters. The benefits of adapter-side indexes are:

- Adapter-side indexes are potentially more space efficient than server-side indexes.

¹¹ Indexed scans cause Informix to ask UFI for rows one-at-a-time. The server thread is blocked until the row is fetched and returned by the adapter. This limits indexed scans to roughly half the frequency at which a process can block and be awakened.

¹² assuming, of course, that there is a composite index on the columns `red` and `blue`.

- An adapter-side index can be created on a UFI table that has several underlying data files.
- Adapter-side indexes don't have the "[4 billion row](#)" limit that server side indexes do.
- Adapter-side indexes are quite fast because they minimize communication between the Informix server and an adapter.
- The user has complete control over which (if any) adapter-side index is used in a query.

The drawbacks of adapter-side indexes are:

- Adapter-side indexes are invisible to the Informix server's optimizer/planner, so it may not generate the most optimal plan in the case of joins.
- Currently, only one adapter-side index can be used per table per query.

The following steps are needed to use adapter-side indexes:

- 1) There must be a file called `$INFORMIXDIR/etc/ufi_index_dir.txt` which contains a single line, the absolute path of the directory that will hold adapter-side indexes for a particular Informix server. This directory needs to have read/write/execute permissions for all users.
- 2) The table to be indexed needs to have an `lvarchar` column mapped to the special locator "[ufi_index](#)". Typically, this column is called `ufi_index`.

- 3) The adapter-side index (or indexes) are created using the function

```
ufi_create_index('tablename', 'column1,column2,...')
```

- 4) The column that is mapped to the `ufi_index` locator (column `ufi_index` in the example below) must be matched against a literal string that names the index (an index's name is the list of columns). For example,

```
SELECT * FROM satellitexix
WHERE ufi_index = 'red,blue' AND red > 49 AND red < 55 AND
blue > 99 AND blue < 105;
```

Non-Adapter-Specific Locators

Generally speaking, locator formats are adapter-specific – e.g., the locators used with an HDF5 file will look different from those that are used with a CSV or NetCDF file. However, there are some locators that are common to all adapter types.

“alias” locator

A special locator called `alias` results in a column whose value is taken from the `alias` column from the [UFI_FILES](#) table. This feature is intended to give users the ability to restrict queries to a particular set of files. For example, suppose we have a column called, say, `dataset` in our `satellitepix` UFI table, and we associate the `alias` locator with this column:

```
execute procedure UFI_add_column(  
'satellitepix', 'dataset', 'alias');
```

Suppose that we associate several files with this UFI table, as follows:

```
execute procedure UFI_add_file('satellitepix', 'ds001',  
'\opt/data/sat303.h5');  
execute procedure UFI_add_file('satellitepix', 'ds002',  
'\opt/data/sat302.h5');  
...
```

Then we can use the following SQL `SELECT` statement to restrict data to just that coming from the file `\opt/data/sat302.h5`.

```
SELECT x, y, red, green, blue FROM satellitepix  
WHERE dataset='ds002';
```

“ufi_index” locator

When we use [adapter-side indexing](#), the table to be indexed needs to have an `lvarchar` column mapped to the special locator called `ufi_index`. Typically, this column is called `ufi_index`, but it doesn't need to be.

UFI Adapters

UFI has two complementary parts:

- 1) a set of VTI DataBlade routines that run inside the database server process, and
- 2) a set of external [interpretation](#)-specific adapter programs that run in separate processes outside the database server (but on the same machine as the database server).

This design is analogous to the CGI gateway supported by web servers, where the web server itself is separated from programs that handle particular types of `httpd` requests. The UFI design has the expected implications:

- The VTI DataBlade routines are much simpler than they would be if we had embedded the file-type-specific logic in the database server, and hence more maintainable.
- Most of the complicated logic is located in the external “adapter” programs, referred to in this document as “adapter” programs or simply “adapters”. External programs are much easier to develop and maintain than database user defined routines (UDRs), particularly when legacy code is involved. Database server code must follow strict rules with regard to memory allocation, file system interaction, etc. Modifying legacy code to conform to these rules can involve a significant amount of work.
- The extensibility of this approach naturally attracts a community effort to write additional adapter programs for the various specialized file formats.

UFI currently comes packaged with the following adapters:

Adapter	Description
CSV	for comma-separated-value files
DBF	for dBase files, such as those used for ESRI shapefiles
NetCDF	for NetCDF files
HDF5	for HDF5 files
GDAL	for file types supported by the Geospatial Data Abstraction Library (GDAL)

Adapter Frameworks

Writers of UFI adapters can choose between two frameworks in writing the adapters: the *Generic Framework* and the *Grid Framework*. Each of these two frameworks provides a set of stub functions which the adapter developer is expected to implement. The Generic Framework makes no assumptions about the data and leaves it up to the adapter developer to organize it. The Grid Framework, on the other hand, assumes the data fits a rectangular grid (or jagged/ragged grid) and provides a great deal of support that is specific to rectangular grids (with regular or jagged/ragged dimensions).

The specific details of how to write an adapter are beyond the scope of this manual¹³, since this manual is for adapter *users*, not adapter *writers*. However, it's important to understand which framework was used in developing a particular adapter, since this can affect how you use it.

Adapter	Framework Used
CSV	Generic
DBF	Grid
NetCDF	Grid
HDF5	Grid
GDAL	Grid

¹³ If you are interested in creating an adapter for a particular file format, please contact BCSInfo@barrodale.com.

CSV Adapter

Locator Format

The locator for a column in a CSV file is simply the string “col” followed by the column position number, starting at 0 (e.g., col0, col1, col2, etc.) See the [example](#) provided earlier (page 22).

Other Notes



Note that the CSV adapter is written using the [Generic Framework](#). Hence the functions [UFI mark ordered](#) and [UFI clear ordered](#) cannot be used.

DBF Adapter

Locator Format

The locator for an attribute in a DBF file is simply the name of the attribute.
The number of rows in the DBF file can be addressed by the locator `'#rows'`.

GDAL Adapter

The GDAL Adapter is based on the open source [GDAL](#) (Geospatial Data Abstraction) library, a library that provides access to a variety of file formats that support raster images. The library (and hence the adapter) is limited to those files that fit within the GDAL model. Before trying to access a file of a new structure via the GDAL adapter, we suggest examining the file with the `gdalinfo` tool that is included with GDAL; if the `gdalinfo` tool can read the file successfully then the GDAL Adapter should as well.

The GDAL Data Model

GDAL models a dataset as having the following components:

- a 2D spatial resolution and a planar projection.
- metadata – a list of key-value pairs associated with the file as a whole.
- an optional set of ground control points, relating one or more positions on the raster to georeferenced coordinates. Each ground control point is a tuple containing an id, an information string, a pixel column, line number, geographic x, geographic y, and geographic z.
- An ordered set of unnamed raster bands. Each raster band is a 2D array of a scalar numeric type or a complex (real + imaginary) type. Currently, only scalar types are supported. Associated with each raster band there can be:
 - metadata – a list of key-value pairs describing the specific band;
 - A raster attribute table – a relational database-like table whose columns are of type integer, double, or string. Raster attribute tables are sometimes used for color maps;
 - several other attributes, e.g., an optional offset and scale for transforming raster values into meaningful values (i.e., translate height to meters), an optional offset and scale for transforming raster values into meaningful values (i.e., translate height to meters), an optional minimum and maximum value, etc.

See the *Data Model* description on the [GDAL website](#) for more information.

Locators supported by the GDAL Adapter

File constants:

Locator	Reference to
<code>file.projection_ref</code>	the spatial reference text for the

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

	projection used by the data.
<code>file.raster_x_size</code>	the number of pixels per line in a band – an integer.
<code>file.raster_y_size</code>	the number of lines per band. An integer.
<code>file.band_count</code>	the number of bands - an integer.
<code>file.geotransform_coefficients</code>	<p>the six element affine transformation used to map pixel and line positions in a raster image to a geographic location. The value is returned as a single string of the form</p> <p>ROW(coef[1],coef[2], coef[3], coef[4],coef[5],coef[6])</p> <p>where <code>coef[*]</code> are floating point values. The interpretation of the coefficients is:</p> $\text{geographicX} = \text{coef}[1] + P * \text{coef}[2] + L * \text{coef}[3];$ $\text{geographicY} = \text{coef}[4] + P * \text{coef}[5] + L * \text{coef}[6];$ <p>where <code>L</code> is the line and <code>P</code> is the pixel value. The top left corner of the top left pixel has <code>L = P = 0</code>. The bottom right corner of the bottom right pixel has</p> <p><code>L=file.raster_y_size</code> and <code>P=file.raster_x_size</code>.</p> <p>Note: the geotransform coefficients won't typically be needed by applications as geographic coordinates are available as well.</p>
<code>file.gcps_count</code>	The number of ground control points – an integer.
<code>file.gcps</code>	a one dimensional array of ground control point values, also known as

	<p>the GCP list. Each point is represented as a string of the form:</p> <pre>ROW(id,info,pixel,line,geox,geoy,geoz)</pre> <p>where <code>id</code> and <code>info</code> are single-quoted strings, and the remaining fields are floating point values.</p>
<p><code>file.metadata.key_name[(format)]</code></p>	<p>The value of one of the key-value pairs associated with the file. The values are stored as strings, but can be mapped to other types if they have an appropriate format. The optional format can be one of the following:</p> <ul style="list-style-type: none"> • days_utc – for datetimes expressed as days since 1970-1-1 GMT. • seconds_utc – for datetimes expressed as seconds since 1970-1-1 GMT. This the default format for datetime columns. • mseconds_utc – for datetimes expressed as milliseconds since 1970-1-1 GMT. • useconds_utc – for datetimes expressed as microseconds since 1970-1-1 GMT. • for datetimes, a format string as accepted by the Linux <code>strptime</code> function. By default, the time zone is assumed to be GMT . A timezone can be specified in the format string¹⁴ with the suffix

¹⁴ Typically, files that come from the same source have times that are expressed in the same timezone. Technically, it's less error prone to have the UFI user describe the timezone in the format line than try to parse it from the data file, assuming it is present.

	<p>%TZ=[+]<i>hours:minutes</i> .</p> <ul style="list-style-type: none"> • one of days, hours, minutes, seconds, mseconds, useconds - for intervals. The last two denote milliseconds and microseconds respectively. <p>For example:</p> <ul style="list-style-type: none"> • seconds_utc • %Y-%m-%d %H:%M:%S - year-month-day_of_month hour:minute:second GMT. • %Y-%m-%d %H:%M:%S%TZ=-8:00 - year-month-day_of_month hour:minute:second PST. • %D %T - day_of_month/month/year hour:minute:second GMT. • days
--	---

Dimensional indexes

Locator	Reference to
<code>index.raster</code>	the current band number (1.. <i>number_of_bands</i>) – an integer.
<code>index.y</code>	the current line of a raster image (1 .. <i>number_of_lines</i>) – an integer.
<code>index.x</code>	the column of a raster image (1.. <i>num_pixels_per_line</i>) – an integer.
<code>index.ratrow</code>	the current row of a raster attribute table – an integer.
<code>index.gcpro</code>	the current row in the ground control point list – an integer.

Geographic Coordinates

Locator	Reference to
<code>geo.x</code>	the X coordinate of the center of the current pixel in a raster band.
<code>geo.y</code>	the Y coordinate of the center of the current pixel in a raster band.

Band Values

Locator	Reference to
band	all bands in the file.
band_i	the <i>i</i> 'th band in the file, where <i>i</i> starts at 1, e.g., band 1, band 23, etc.
<code>band_match(key, regular_expression)</code>	<p>the set of bands that have a meta-data item that matches the specified regular expression. The regular expression should be double quoted. Different columns (in the same UFI table) can reference different sets of bands, as long as each set has the same number of bands in it. The regular expression follows the POSIX Extended Regular Expression syntax with the addition that “^” and “\$” are prepended and appended respectively to the pattern so that the entire pattern is forced to match the entire string.</p> <p>Examples:</p> <pre>band_match(GRIB_ELEMENT, "UGRD") band_match(GRIB_ELEMENT, ".*GRD")</pre>
<code>band_reference.metadata.key_name[(time_format)]</code>	<p>the metadata value associated with keys of bands. The <code>band_reference</code> is either <code>band</code>, <code>band_i</code>, or <code>band_match(key, regular_expression)</code>. For example:</p> <pre>band_match(GRIB_ELEMENT, "UGRD"). metadata.GRIB_REF_TIME(seconds_utc) band_match(GRIB_ELEMENT, "UGRD"). metadata.GRIB_FORECAST_SECONDS(seconds)</pre>
<code>band_reference.rat.rat_column_name:</code>	<p>the value in a raster attribute table. The <code>band_reference</code> is either <code>band</code>, <code>band_i</code>, or <code>band_match(key, regular_expression)</code>. For example:</p> <pre>band_1.rat.green band_1.rat.blue</pre>

Note: the GDAL equivalent of NetCDF's `fillvalue` attribute is a property called the `nodata` value. The GDAL adapter will translate `nodata` values found

in a band to `null`. The adapter will also take into account the [scale and offset properties](#).

Supporting Additional Raster Formats

The GDAL adapter uses a shared GDAL library (`libgdal.so`) which it looks for in the `$INFORMIXDIR/extend/UniversalFileInterface.1.0.0.0/adapters/gdal_libs` directory. To support a new raster format, download a copy of the GDAL source code from <http://www.gdal.org>, add your own driver, and replace the existing `libgdal.so` file¹⁵.

¹⁵ However, it should be pointed out that it's likely easier to support a new format directly, using the UFI Adapter Library to create a new adapter of your own. If you are interested in creating an adapter for a particular file format, please contact BCSInfo@barrodale.com.

HDF5 Adapter

HDF5 Terminology (from [HDF User's Guide](#))

Dataset	A dataset is a multidimensional array of data elements, together with supporting metadata.
Group	A group is a structure containing instances of zero or more other groups or datasets, together with supporting metadata.
Attribute	An attribute is a string of the form “name = value”, attached to an object (group or dataset).
Path Name	A path name is a string of components separated by slashes (/). Each component is the name of a hard or soft link which points to an object in the file. The slash not only separates the components, but indicates their hierarchical relationship; the component indicated by the link name following a slash is always a member of the component indicated by the link name preceding that slash.
Hyperslab	A hyperslab is a selection of elements from a hyper rectangle. An HDF5 hyperslab is a rectangular pattern defined by four arrays. The four arrays are described in the next four rows of this table
Hyperslab Offset	The starting location for the hyperslab.
Hyperslab Stride	The number of elements to separate each element or block to be selected.
Hyperslab Count	The number of elements or blocks to select along each dimension.
Hyperslab Block	The size of the block selected from the dataspace.

Locator Format

The [locator](#) format used by UFI for HDF5 file [objects](#) is an extension of the [Path Names](#) format used within HDF5 itself. The extension allows UFI to *iterate* over the elements of datasets in a logical way.

Consider the following simple HDF file:

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

```

GROUP "/" {
  GROUP "m" {
    ATTRIBUTE "attr" {
      DATATYPE HST_STRING { STRSIZE 17; }
      DATA { "string attribute" }
    }
  }
  GROUP "x" {
    DATASET "y" {
      DATATYPE H5T_COMPOUND {
        H5T_STD_I32BE "z";
        H5T_STD_I32BE "p";
      }
      DATASPACE SIMPLE { (3, 3)/(3,3) }
      DATA {
        {{1,2},{3,4},{5,6}},
        {{7,8},{9,10},{11,12}},
        {{13,14},{15,16},{17,18}}
      }
    }
  }
}

```

The following are some examples of locators for this file:

Locator	Reference to
/m/ATTR:attr	the value of attr, i.e., "string attribute".
/x/y[*,*]/z	z in different elements of array y in each iteration.
/x/y[\$,*]	the first index of array y in a particular iteration.
/x/y[stride,*]	the hyperslab stride of the first index of array y.
/x/y[blocking,*]	the hyperslab block of the first index of array y in a particular iteration.
/x/y[* ,idval:\$]	the second index of y in a particular iteration. The literal value "idval" is associated with the dimension so it can be linked to a dimension of another variable. We could have used any alpha-numeric style identifier instead of "idval".

<code>/x/y[* ,idval:stride]</code>	the hyperslab stride of the second index of array y .
<code>/x/y[* ,smith:blocking]</code>	the hyperslab block of the second index of array y in a particular iteration.

The `stride` and `blocking` locator keywords allow subsampling to be performed. They are set by `WHERE` clause terms referencing table columns that are mapped to `stride` or `blocking` locators. For example, consider a table `xytab` with the following *column* → *locator* mappings defined against the HDF5 file shown [above](#):

- `zvalues` → `'/x/y[* ,*]/z'`
- `indexmajor` → `'/x/y[$,*]'`
- `indexminor` → `'/x/y[$,*]'`
- `stridemajor` → `'/x/y/[stride,*]'`
- `blockingmajor` → `'/x/y/[blocking,*]'`

This table could be sampled 2 rows out of 3 with the following query:

```
SELECT indexmajor, indexminor, zvalues FROM xytab WHERE
stridemajor = 3 AND blockingmajor = 2;
```

producing the following 6 results:

```
0,0,1
0,1,3
0,2,5
1,0,7
1,1,9
1,2,11
```

By default, the [hyperslab stride](#) and [block](#) have values equal to a dimension's length. If a hyperslab stride is set to less than the dimension's length, but the hyperslab block isn't explicitly set, the hyperslab block is explicitly set to 1. The `stride` and `blocking` terms cannot be applied to variable length dimensions.

Group Matching in a Locator

HDF5 producers have been known to use groups in place of arrays to get hierarchically structured keys. For example, consider the following HDF5 file:

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

```
GROUP "/" {
  GROUP "MONTH1" {
    GROUP "DAY1" {
      ATTRIBUTE "odometer" {
        DATATYPE H5T_STD_I32BE
        DATA { 100 }
      }
    }
    GROUP "DAY2" {
      ATTRIBUTE "odometer" {
        DATATYPE H5T_STD_I32BE
        DATA { 120 }
      }
    }
    GROUP "DAY3" {
      ATTRIBUTE "odometer" {
        DATATYPE H5T_STD_I32BE
        DATA { 130 }
      }
    }
    GROUP "DAY4" {
      ATTRIBUTE "odometer" {
        DATATYPE H5T_STD_I32BE
        DATA { 151 }
      }
    }
  }
  GROUP "MONTH2" {
    GROUP "DAY1" {
      ATTRIBUTE "odometer" {
        DATATYPE H5T_STD_I32BE
        DATA { 170 }
      }
    }
    GROUP "DAY2" {
      ATTRIBUTE "odometer" {
        DATATYPE H5T_STD_I32BE
        DATA { 180 }
      }
    }
  }
}
```

A user may want to view this file as a table with three columns: month, day, and odometer_setting. This can be done using a special notation for matching groups in the locator. The notation has the form

matches(regular_expression_string)

and is either followed by a pseudo primitive called **name** or a pseudo group called **matched**. The *regular_expression_string* uses the POSIX Extended Regular Expression syntax (described [here](#)) with the following additions:

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

- The string “*” is translated to the pattern “^.*\$”, which matches any string.
- A “^” and “\$” are prepended and appended respectively to the pattern so that the entire pattern is forced to match the entire string.

The `name` pseudo primitive represents the text matched by the regular expression. The `matched` pseudo group represents the entire group that was matched.

Possible locators that could be used to associate the `month`, `day`, and `odometer_setting` columns with HDF5 file components might be, respectively,

```
/matches ("MONTH[0-9]+") /name  
/matches ("MONTH[0-9]+") /matched/matches ("DAY[0-9]+") /name  
/matches ("MONTH[0-9]+") /matched/matches ("DAY[0-  
9]+") /matched/odometer
```

There is a program, called `regexCheck`, included in

```
$INFORMIXDIR/extend/UniversalFileInterface.1.0.0.0/bin
```

that can be used to determine which strings will be matched by a particular regular expression. The regular expression is supplied on the command line, and the input strings read from `stdin`. Each string is echoed back to `stdout`, along with the text “ -matched” or “-notmatched”. The regular expressions should be quoted to ensure that the shell does not expand them.

The following is an example of the use of `regexCheck`:

```
regexCheck '(JAN|FEB)[0-9]+' <<XXX  
JAN  
MAR3  
FEB0  
JAN8  
FEB101010  
XXX
```

produces the following output:

```
JAN -notmatched  
MAR3 -notmatched  
FEB0 -matched  
JAN8 -matched  
FEB101010 -matched
```

NetCDF Adapter

NetCDF Terminology (from the [Unidata site](#))

dimension	A dimension may be used to represent a real physical dimension, for example, time, latitude, longitude, or height. A dimension might also be used to index other quantities, for example station or model-run-number. A netCDF dimension has both a name and a length.
variable	The basic unit of named data in a netCDF dataset is a variable. When a variable is defined, its shape is specified as a list of dimensions. A variable represents an array of values of the same type. A scalar value is treated as a 0-dimensional array. A variable has a name, a data type, and a shape described by its list of dimensions specified when the variable is created. A variable may also have associated attributes, which may be added, deleted or changed after the variable is created.
coordinate variable	A variable with the same name as a dimension is called a <i>coordinate variable</i> . It typically defines a physical coordinate corresponding to that dimension.
attribute	Attributes are used to store data about the data (ancillary data or metadata), similar in many ways to the information stored in data dictionaries and schema in conventional database systems. Most attributes provide information about a specific variable. These are identified by the name (or ID) of that variable, together with the name of the attribute.
global attribute	Some attributes provide information about the dataset as a whole and are called global attributes.
CDL	Common Data form Language: a human-readable text representation of netCDF data.

Locator Format

The [locator](#) for a NetCDF [dimension](#) that does not have a [coordinate variable](#) of the same name is the name of the dimension. If a dimension has the same name as its coordinate variable then an index to the dimension can be referred to by

prepending the name with a '\$' . The length of the variable can be referred to by prepending the name with a '#' .

The locator for any [variable](#) is the name of the variable.

The locator for the [attribute](#) *attribute_name* of a particular [variable](#) *variable_name* is *variable_name.attribute_name* .

The locator for a [global attribute](#) *attribute_name* is **nc_globals**.*attribute_name* or **nc_globals:***attribute_name*.

Example

As a concrete example, consider the following NetCDF file (expressed here in [CDL](#)):

```
netcdf example {
  dimensions:
    episode = 10 ;
    latitude = 5 ;
  variables:
    double latitude(latitude) ;
        latitude:units = "degrees north of the equator";
    int eventsType(latitude, episode) ;

  // global attributes:
        :Conventions = "none" ;
  data:

    latitude = -40, -20, 0, 20, 40 ;

  eventsType =
    101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
    111, 112, 113, 114, 115, 126, 197, 200, 201, 210,
    201, 102, 203, 204, 305, 306, 107, 108, 109, 220,
    301, 302, 303, 304, 305, 306, 307, 308, 309, 230,
    401, 402, 403, 404, 405, 405, 407, 408, 409, 430 ;
}
```

The following locators could be used with this file:

Locator	Reference to
episode or \$episode	The index to the episode dimension.
#episode	The size of the episode dimension.
latitude	The latitude coordinate variable.

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

\$latitude	The index into the latitude dimension.
#latitude	The size of the latitude dimension.
latitude:units	The units attribute of the latitude variable
eventsType	The eventsType variable
nc_globals:Conventions	The Conventions global attribute

Example

```

CREATE TABLE example_meta_template(
    episode_length INTEGER,
    latitude_length INTEGER,
    conventions      LVARCHAR,
    latitude_units   LVARCHAR
) ;

EXECUTE PROCEDURE ufi_make_managed('example_meta', 'netcdf',
    'example_meta_template');
EXECUTE PROCEDURE ufi_add_column('example_meta',
    'episode_length', '#episode');
EXECUTE PROCEDURE ufi_add_column('example_meta',
    'latitude_length', '#latitude');
EXECUTE PROCEDURE ufi_add_column('example_meta', 'conventions',
    'nc_globals:Conventions');
EXECUTE PROCEDURE ufi_add_column('example_meta',
    'latitude_units', 'latitude:units');
EXECUTE PROCEDURE ufi_add_file('example_meta', 'example_meta',
    '/opt/data/example.nc');

SELECT * FROM example_meta;

episode_length  10
latitude_length  5
conventions      none
latitude_units   degrees north of the equator

CREATE TABLE example_template(
    episode_i integer,
    latitude_i integer,
    latitude    double precision,
    eventtype integer
) ;

EXECUTE PROCEDURE ufi_make_managed('example', 'netcdf',

```

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

```
        'example_template');  
EXECUTE PROCEDURE ufi_add_column('example', 'episode_i',  
    '$episode');  
EXECUTE PROCEDURE ufi_add_column('example', 'latitude_i',  
    '$latitude');  
EXECUTE PROCEDURE ufi_add_column('example', 'latitude',  
    'latitude');  
EXECUTE PROCEDURE ufi_add_column('example', 'eventtype',  
    'eventsType');  
EXECUTE PROCEDURE ufi_add_file('example', 'example_data',  
    '/opt/data/example.nc');  
SELECT * FROM example WHERE latitude = 20 AND episode_i < 5;
```

episode_i	latitude_i	latitude	eventtype
1	4	20.000000000000	301
2	4	20.000000000000	302
3	4	20.000000000000	303
4	4	20.000000000000	304

Troubleshooting

Most errors detected by a UFI adapter are properly propagated back to the Informix server, whereupon they terminate a query and generate an error message. In some rare circumstances, this can fail. Usually, in such cases, you can halt the query cleanly by creating a file called `/tmp/ufi_halt`, and then deleting the file afterwards (if UFI didn't manage to remove it itself).

Currently, adapters copy their instructions (as they read them) into a file called `/tmp/ufi_adapter.log`. If an adapter crashes, it typically leaves a message in this log file.

Glossary

adapter program: An adapter program (or simply “adapter”) is an external (i.e., external to the DBMS server) program tasked with reading a particular type of file in response to requests from the core UFI server component.

array: A multidimensional array (vector, matrix, 4D cube,...) without metadata other than array dimensions. In general, we mean ragged/jagged arrays and rectangular arrays.

dataset: A logical group of files that all have the same structure. For example, a dataset may be a group of files, each recording the precipitation on a different day.

group: A set of named [objects](#) of possibly different types.

object: An object may be an instance of a primitive or compound datatype (integer, floating point value, or string), or a multidimensional array of objects of a common type.

locator: A textual reference to an [object](#) within a file, used to identify the [object](#) (or group of [objects](#), as locators can include wildcards). The exact format of a locator depends on the external program used to interpret the associated file.

interpretation: Typically, files can be interpreted at different levels. At the lowest level, there is the file type (tiff, NetCDF, HDF5, etc.). Above the level of file type, there may be conventions that add meaning to how the data in a file is interpreted (geotiff, COARDS, HDF-EOS, CF-1.0, etc.). Above this, applications may synthesize additional information (such as bounding boxes for indexes, conversions to different units, etc). In general, an interpretation combines all three of these: it is a conversion from a particular class of files to tabular data.

Appendix 1: Performing Table Joins With UFI Tables

For many types of queries UFI can perform direct access within a netCDF or HDF5 file, significantly speeding up these queries. Consider, for example, the UFI table generated by the following SQL:

```
> create table windsu_template(  
    lon double precision,  
    lat double precision,  
    u double precision  
);  
> execute procedure ufi_make_managed('windsu', 'netcdf',  
    'windsu_template');  
> execute procedure ufi_add_column('windsu', 'lon', 'g5_lon_1');  
> execute procedure ufi_add_column('windsu', 'lat', 'g5_lat_0');  
> execute procedure ufi_add_column('windsu', 'u',  
> execute procedure ufi_add_file('windsu', 'windsu',  
    netcdfFileName);
```

If `G5_LON_1` is a netCDF dimension variable, then a query like

```
SELECT u FROM windsu WHERE lon between lowval and highval;
```

can be performed much faster than if `G5_LON_1` were not a dimension variable. In the former case, UFI can use the `G5_LON_1` dimension array to figure out where in the file the requested `u` values are, and then use direct access to fetch them. In the latter case, however, UFI will need to perform a full file scan on the netCDF file.

If `lowval` and `highval` in the query above are values from a column in another table, e.g.,

```
SELECT u FROM windsu w, bounds b WHERE  
    lon BETWEEN b.low AND b.high AND b.keyval BETWEEN ... and ...;
```

then special care must be taken to ensure that UFI has the information it needs to do direct access.

The first requirement is that an “UPDATE STATISTICS” command be issued for a UFI table before its first appearance in a `SELECT` statement.

If `bounds` is a conventional table (and `windsu` is a UFI table) and there is an index on `keyval` (or if `bounds` is sufficiently small that an index is not necessary), then the query directive in the following `SELECT` statement will result in a fast execution:

**THE UNIVERSAL FILE INTERFACE (UFI)
USER'S GUIDE**

```
SELECT {+ORDERED, FULL(w)} u
FROM bounds b, windsu w
WHERE lon BETWEEN b.low AND b.high AND
      b.keyval between ... AND ...;
```

In the SQL above, we've switched the order of the tables in the FROM clause and specified the ORDERED directive. This will result in a fast (index) scan of bounds, and for each row in bounds that is returned, a VTI scan of windsu will be performed. We specify the FULL directive for windsu since we want UFI to have control over how to access this file (and hence be able to exploit the dimension arrays to perform direct access)¹⁶.

The same query directive will work if bounds is a UFI table as long as keyval is a dimension variable (for netCDF files) or an [array](#) index in an HDF5 file.

¹⁶ That a specification of the FULL directive causes a direct access by UFI might seem confusing. The FULL directive is used by the Informix engine to decide how to process the table; basically it results in UFI being given total control over how to process the table (and UFI always uses direct access if it can). Without the FULL directive, the Informix engine might instead try to perform an autoindex on the UFI table, resulting in very slow performance since generating the index actually would result in a full file scan.

Appendix 2: The UFI Table Schema

<pre>CREATE TABLE ufi_adapters (adapterpath lvarchar, interpretation varchar(20));</pre>	<table border="1"> <thead> <tr> <th colspan="2">adapterpath</th> <th>interpretation</th> </tr> </thead> <tbody> <tr> <td>\$UFI_ADAPTERS/hdf5Adapter</td> <td></td> <td>hdf5</td> </tr> <tr> <td>\$UFI_ADAPTERS/netcdfAdapter</td> <td></td> <td>netcdf</td> </tr> </tbody> </table>	adapterpath		interpretation	\$UFI_ADAPTERS/hdf5Adapter		hdf5	\$UFI_ADAPTERS/netcdfAdapter		netcdf																																	
adapterpath		interpretation																																									
\$UFI_ADAPTERS/hdf5Adapter		hdf5																																									
\$UFI_ADAPTERS/netcdfAdapter		netcdf																																									
<pre>CREATE TABLE ufi_datatypes (tabid integer, interpretation varchar(20));</pre>	<table border="1"> <thead> <tr> <th>tabid</th> <th colspan="2">filetype</th> </tr> </thead> <tbody> <tr> <td>340</td> <td></td> <td>hdf5</td> </tr> <tr> <td>341</td> <td></td> <td>hdf5</td> </tr> <tr> <td>342</td> <td></td> <td>netcdf</td> </tr> </tbody> </table>	tabid	filetype		340		hdf5	341		hdf5	342		netcdf																														
tabid	filetype																																										
340		hdf5																																									
341		hdf5																																									
342		netcdf																																									
<pre>CREATE TABLE ufi_files (tabid integer, alias lvarchar, filepath lvarchar);</pre>	<table border="1"> <thead> <tr> <th>tabid</th> <th>alias</th> <th>datasetpath</th> </tr> </thead> <tbody> <tr> <td>340</td> <td>ds01</td> <td>/opt/data/sat303.hdf</td> </tr> <tr> <td>340</td> <td>ds02</td> <td>/opt/data/sat302.hdf</td> </tr> <tr> <td>341</td> <td>ds03</td> <td>/opt/data/sat404.hdf</td> </tr> <tr> <td>342</td> <td>fish1</td> <td>/opt/data/2008Precip.nc</td> </tr> </tbody> </table>	tabid	alias	datasetpath	340	ds01	/opt/data/sat303.hdf	340	ds02	/opt/data/sat302.hdf	341	ds03	/opt/data/sat404.hdf	342	fish1	/opt/data/2008Precip.nc																											
tabid	alias	datasetpath																																									
340	ds01	/opt/data/sat303.hdf																																									
340	ds02	/opt/data/sat302.hdf																																									
341	ds03	/opt/data/sat404.hdf																																									
342	fish1	/opt/data/2008Precip.nc																																									
<pre>create table ufi_columns (tabid integer, colname varchar(80), locator lvarchar);</pre> <p>Note: the locator field's format is specific to particular adapter programs.</p>	<table border="1"> <thead> <tr> <th>tabid</th> <th>colname</th> <th>locator</th> </tr> </thead> <tbody> <tr> <td>340</td> <td>x</td> <td>/image/pixel[*,\$]</td> </tr> <tr> <td>340</td> <td>y</td> <td>/image/pixel[\$,*]</td> </tr> <tr> <td>340</td> <td>red</td> <td>/image/pixel[*,*]/red</td> </tr> <tr> <td>340</td> <td>green</td> <td>/image/pixel[*,*]/green</td> </tr> <tr> <td>340</td> <td>blue</td> <td>/image/pixel[*,*]/blue</td> </tr> <tr> <td>341</td> <td>x</td> <td>/image/plane[0]/val[*,\$]</td> </tr> <tr> <td>341</td> <td>y</td> <td>/image/plane[0]/val[\$,*]</td> </tr> <tr> <td>341</td> <td>red</td> <td>/image/plane[0]/val[*,*]</td> </tr> <tr> <td>341</td> <td>green</td> <td>/image/plane[1]/val[*,*]</td> </tr> <tr> <td>341</td> <td>blue</td> <td>/image/plane[2]/val[*,*]</td> </tr> <tr> <td>341</td> <td>dataset</td> <td>alias</td> </tr> <tr> <td>342</td> <td>timestamp</td> <td>timestamp</td> </tr> <tr> <td>342</td> <td>level</td> <td>rainlevel</td> </tr> </tbody> </table>	tabid	colname	locator	340	x	/image/pixel[*,\$]	340	y	/image/pixel[\$,*]	340	red	/image/pixel[*,*]/red	340	green	/image/pixel[*,*]/green	340	blue	/image/pixel[*,*]/blue	341	x	/image/plane[0]/val[*,\$]	341	y	/image/plane[0]/val[\$,*]	341	red	/image/plane[0]/val[*,*]	341	green	/image/plane[1]/val[*,*]	341	blue	/image/plane[2]/val[*,*]	341	dataset	alias	342	timestamp	timestamp	342	level	rainlevel
tabid	colname	locator																																									
340	x	/image/pixel[*,\$]																																									
340	y	/image/pixel[\$,*]																																									
340	red	/image/pixel[*,*]/red																																									
340	green	/image/pixel[*,*]/green																																									
340	blue	/image/pixel[*,*]/blue																																									
341	x	/image/plane[0]/val[*,\$]																																									
341	y	/image/plane[0]/val[\$,*]																																									
341	red	/image/plane[0]/val[*,*]																																									
341	green	/image/plane[1]/val[*,*]																																									
341	blue	/image/plane[2]/val[*,*]																																									
341	dataset	alias																																									
342	timestamp	timestamp																																									
342	level	rainlevel																																									